Matter Specification

Version 1.0

Document:	22-22349-001_Matter-1.0-Core-Specification.pdf
	September 28, 2022
Sponsored by:	Connectivity Standards Alliance
Accepted by:	This document has been accepted for release by the Connectivity Standards Alliance Board of Directors on **September 28, 2022**
Abstract:	The Matter specification defines fundamental requirements to enable an interoperable application layer solution for smart home devices over the Internet Protocol.
Keywords:	Referenced in Chapter 1.

Copyright © 2022 Connectivity Standards Alliance, Inc.
508 Second Street, Suite 109B Davis, CA 95616 - USA
www.csa-iot.org
All rights reserved.

Permission is granted to members of the Connectivity Standards Alliance to reproduce this document for their own use or the use of other Connectivity Standards Alliance members only, provided this notice is included. All other rights reserved. Duplication for sale, or for commercial or for-profit use is strictly prohibited without the prior written consent of the Connectivity Standards Alliance.

Matter Specification

Version 1.0, 2022-09-23 12:33:19 -0700: Approved

Copyright Notice, License and Disclaimer

Copyright © Connectivity Standards Alliance (2022). All Rights Reserved. The information within this document is the property of the Connectivity Standards Alliance and its use and disclosure are restricted, except as expressly set forth herein.

Connectivity Standards Alliance hereby grants you a fully-paid, non-exclusive, nontransferable, worldwide, limited and revocable license (without the right to sublicense), under Connectivity Stan! dards AllianceÕs applicable copyright rights, to view, download, save, reproduce and use the docu! ment solely for your own internal purposes and in accordance with the terms of the license set forth herein. This license does not authorize you to, and you expressly warrant that you shall not: (a) permit others (outside your organization) to use this document; (b) post or publish this docu! ment; (c) modify, adapt, translate, or otherwise change this document in any manner or create any derivative work based on this document; (d) remove or modify any notice or label on this docu! ment, including this Copyright Notice, License and Disclaimer. The Connectivity Standards Alliance does not grant you any license hereunder other than as expressly stated herein.

Elements of this document may be subject to third party intellectual property rights, including without limitation, patent, copyright or trademark rights, and any such third party may or may not be a member of the Connectivity Standards Alliance. Connectivity Standards Alliance members grant other Connectivity Standards Alliance members certain intellectual property rights as set forth in the Connectivity Standards Alliance IPR Policy. Connectivity Standards Alliance members do not grant you any rights under this license. The Connectivity Standards Alliance is not responsi! ble for, and shall not be held responsible in any manner for, identifying or failing to identify any or all such third party intellectual property rights. Please visit www.csa-iot.org for more information on how to become a member of the Connectivity Standards Alliance.

This document and the information contained herein are provided on an ÒAS ISÓ basis and the Con! nectivity Standards Alliance DISCLAIMS ALL WARRANTIES EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO (A) ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OF THIRD PARTIES (INCLUDING WITHOUT LIMITATION ANY INTELLEC! TUAL PROPERTY RIGHTS INCLUDING PATENT, COPYRIGHT OR TRADEMARK RIGHTS); OR (B) ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE OR NONINFRINGEMENT. IN NO EVENT WILL THE CONNECTIVITY STANDARDS ALLIANCE BE LIABLE FOR ANY LOSS OF PROFITS, LOSS OF BUSINESS, LOSS OF USE OF DATA, INTERRUPTION OF BUSI! NESS, OR FOR ANY OTHER DIRECT, INDIRECT, SPECIAL OR EXEMPLARY, INCIDENTIAL, PUNITIVE OR CONSEQUENTIAL DAMAGES OF ANY KIND, IN CONTRACT OR IN TORT, IN CONNECTION WITH THIS DOCUMENT OR THE INFORMATION CONTAINED HEREIN, EVEN IF ADVISED OF THE POSSI! BILITY OF SUCH LOSS OR DAMAGE.

All company, brand and product names in this document may be trademarks that are the sole prop! erty of their respective owners.

This notice and disclaimer must be included on all copies of this document.

Connectivity Standards Alliance
508 Second Street, Suite 206
Davis, CA 95616, USA

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Participants

Agrawal, Amit	Alexander, Rob	Ananthakrishnan, Krithika	Axelsson, Ulf
Azria, Shana	Bak, Naama	Balducci, Alex	Bao, Yongming
Bartolome, Diego	Bauer-Schwan, Stefan	Beach, Chris	Becker, Markus
Ben, Thomas	Bhetanabottla, Sriram	Bonnell, Corey	C, Rajashree
Carlin, Broderick	Carmel-Veilleux, Ten! nessee	Casallas, Ricardo	Chalmers, Andrew
Chan, Osborn	Chandarana, Janak	Cheshire, Stuart	Chudinov, Adrian
Chupp, Anton	Coppock, Kevin	Cowan, Michael	Cragie, Robert
Crettenand, Alexander	Cuyckens, Thomas	Damle, Makarand	Darling, Don
Decenzo, Chris	Dhayagude, Hrishikesh	Ding, Li-An	Dok, Hrishikesh
Dolan, David	Dong, Kangping	Duda, "ukasz	Dyck, Nathan
Erickson, Grant	Feraru, Eugen	Freeman, Cecille	Fu, Kenneth
Fyall, Ian	Garbus, Mathias Kiel! gast	Garg, Pankaj	Gucea, Doru
Guiheneuf, Robin-Charles	Guo, Jiacheng	Guo, Song	Haefner, Kyle
Hamilton, Ryan	Hanna, John	Hanna, Steve	Haque, Asad
Harris, Will	Heide, Janus	Hernandez-Palomares, Martin	Hoang, Minhhoa
Holbrook, Trevor	Hollebeek, Tim	Houtepen, Rob	Hui, Jonathan
Hui, Li	Jain, Amit	Jain, Ankur	Jandhyala, Chaitanya
Jayakumar, Liju	Johns, Jerry	Josefsen, RenŽ	KY, Suma
Kardous, Mathieu	Kasperczyk, Kamil	Katira, Utsav	Knšrzer, Clemens
Kohr, John	Kommareddi, Naveen	Kontra, Andrew	Kovacic, Lazar
Krawetz, Bryan	Kr—lik, Damian	Kumar, Sandeep	Kumar, Saurabh
Lazar, Alin	Le Tutour, Jean	Lee, Byungjoo	Lepage, Marc
Liang, Deng	Lindeman, Ryan	Litvin, Andrei	Lyu, Rashid
Maes, Timothy	Mamo, Fesseha	Manley, Tom	Mann, Bryan
Mansour, Peter	Margolis, Evgeni	Martinez, Junior	Matignon, Florent
Matosian, Dan	Melo, Sara	Menzopol, Andrei	Moneta, Daniel
Montenegro, Gabriel	Morales, Victor	Morozov, Evgeniy	MŽgevand, Jonathan
Nadathur, Anush	Nicolas, Vivien	Nuyts, Wim	P, Aswathy
Pan, Liam	Pan, Shaofeng	Parausanu, Dragos	Patil, Shubham

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Penven, Jean-Francois	Po, Kevin	Powell, Ken	Pyasi, Madhur
Rempel, David	Rhees, Jon	Rosenberg, Aron	Rozendaal, Leo
Rupp, Michael	S, Sowmya	Sallas, Sal	Sambles, Philip
Sandstedt, Michael	Sarma, Bhaskar	Schiller, Bill	Schoinas, Yannis
Segal, Oren	Sena, Joe	She, Chengqiang	Shreve, Erik
Smirl, Jon	Smith, Bill	Smith, David	Smith, Matt
Soloway, Alan	Son, Jae	Szablowski, Micha#	Szatmary-Ban, Zoltan
Szczodrak, Marcin	Szewczyk, Robert	Trayer, Mark	Turon, Martin
Vauclair, Marc	Verma, Lochan	Wang, David	Wang, Yufeng
Wang, Yunhan	Wei, Qingyun	Weil, Jason	Weinshel, Ben
Weir, Tristan	Williams, Cam	Wood, Justin	Xu, Yakun
Yang, Carol	Zbarsky, Boris	Zgrablic, Leonard	Zang, Mingjie
Zhang, Xili	Zhao, Betty	Zhao, Ru	Zhodzishsky, Victor

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Document Control

The Matter specification is made of individual chapters such as this one. See Chapter 1 for the list of all chapters. References between chapters are made using a X.Y notation where X is the chapter and Y is the sub-section within that chapter. References to external documents are contained in Chapter 1 and are made using [Rn] notation. An update to any of these chapters will be reflected in an update to the source document list below.

Chapter 01$Ñ$Introduction	Document # [./Ch01_Introduction.adoc]
Chapter 02$Ñ$Architecture	Document # [./Ch02_Architecture.adoc]
Chapter 03$Ñ$Cryptographic Primitives	Document # [./Ch03_Cryptography.adoc]
Chapter 04$Ñ$Secure Channel	Document # [./Ch04_Secure_Channel.adoc]
Chapter 05$Ñ$Commissioning	Document # [./Ch05_Commissioning.adoc]
Chapter 06$Ñ$Device Attestation	Document # [./Ch06_Attestation.adoc]
Chapter 07$Ñ$Data Model	Document # [./Ch07_Data_Model.adoc]
Chapter 08$Ñ$Interaction Model	Document # [./Ch08_Interaction_Model.adoc]
Chapter 09$Ñ$System Model	Document # [./Ch09_System_Model.adoc]
Chapter 10$Ñ$Interaction Encoding	Document # [./Ch10_Interaction_Encoding.adoc]
Chapter 11$Ñ$Device Management	Document # [./Ch07_Management.adoc]
Chapter 12$Ñ$Multiple Fabrics	Document # [./Ch09_MultipleAdmins.adoc]
Chapter 13$Ñ$Security Requirements	Document # [./Ch10_Security_Requirements.adoc]

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Revision History

Revision	Date	Details	Editor
01	May 11, 2020	Initial draft	Robert Szewczyk
02	September 23, 2022	Version 1.0	Robert Szewczyk

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Table of Contents

Copyright Notice, License and Disclaimer . . . 1
Participants . . . 3
Document Control . . . 5
Revision History . . . 7
1. Introduction . . . 31
1.1. Scope and Purpose . . . 31
1.2. Acronyms and Abbreviations . . . 31
1.3. Definitions . . . 33
1.4. Standards Terminology Mapping . . . 36
1.5. Conformance Levels . . . 37
1.6. References . . . 37
1.6.1. CSA Reference Documents . . . 38
1.6.2. External Reference Documents . . . 38
1.7. Informative References . . . 42
1.7.1. CSA Reference Documents . . . 42
1.8. Conventions . . . 43
1.8.1. Enumerations and Reserved Values . . . 43
1.8.2. Reserved Bit Fields . . . 43
1.8.3. Number Format . . . 43
1.8.4. Provisional . . . 44
2. Architecture . . . 45
2.1. Overview . . . 45
2.2. Layered Architecture . . . 45
2.3. Network Topology . . . 47
2.3.1. Single network . . . 47
2.3.2. Star network topology . . . 48
2.4. Scoped names . . . 49
2.5. Identifiers . . . 50
2.5.1. Fabric References and Fabric Identifier . . . 50
2.5.2. Vendor Identifier (Vendor ID, VID) . . . 50
2.5.3. Product Identifier (Product ID, PID) . . . 51
2.5.4. Group Identifier (GID) . . . 51
2.5.5. Node Identifier . . . 52
2.5.6. IPv6 Addressing . . . 54
2.6. Device identity . . . 55
2.7. Security . . . 56
2.8. Device Commissioning . . . 56
2.9. Sleepy End Device (SED) . . . 57

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

2.10. Data Model Root 58
2.11. Stack Limits 58
2.11.1. System Model Limits 58
2.11.2. Interaction Model Limits 58
2.12. List of Provisional Items 59
2.12.1. Invoke Multiple Paths 59
2.12.2. EventList Global Attribute 59
2.12.3. Proxy Service 59
2.12.4. Time Synchronization 59
2.12.5. Parameters and Constants 59
3. Cryptographic Primitives 61
3.1. Deterministic Random Bit Generator (DRBG) 61
3.2. True Random Number Generator (TRNG) 62
3.3. Hash function (Hash) 62
3.4. Keyed-Hash Message Authentication Code (HMAC) 63
3.5. Public Key Cryptography 63
3.5.1. Group 64
3.5.2. Key generation 64
3.5.3. Signature and verification 65
3.5.4. ECDH 66
3.5.5. Certificate validation 66
3.5.6. Time and date considerations for certificate path validation 67
3.6. Data Confidentiality and Integrity 68
3.6.1. Generate and encrypt 69
3.6.2. Decrypt and verify 70
3.7. Message privacy 71
3.7.1. Privacy encryption 71
3.7.2. Privacy decryption 72
3.8. Key Derivation Function (KDF) 72
3.9. Password-Based Key Derivation Function (PBKDF) 74
3.10. Password-Authenticated Key Exchange (PAKE) 75
3.10.1. Computation of pA 77
3.10.2. Computation of pB 77
3.10.3. Computation of transcript TT 78
3.10.4. Computation of cA, cB and Ke 78
4. Secure Channel 79
4.1. General Description 79
4.1.1. Messages 79
4.2. IPv6 Reachability 80
4.2.1. Stub Router Behavior 81
4.2.2. Matter Node Behavior 81

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

4.3. Discovery 81
4.3.1. Commissionable Node Discovery 83
4.3.2. Operational Discovery 99
4.3.3. Commissioner Discovery 103
4.3.4. Common TXT Key/Value Pairs 106
4.4. Message Frame Format 107
4.4.1. Message Header Field Descriptions 108
4.4.2. Message Footer Field Descriptions 111
4.4.3. Protocol Header Field Descriptions 112
4.4.4. Message Size Requirements 114
4.5. Message Counters 114
4.5.1. Message Counter Types 114
4.5.2. Secure Session Message Counters 116
4.5.3. Message Counters as Encryption Nonces 116
4.5.4. Replay Prevention and Duplicate Message Detection 117
4.5.5. Counter Processing of Outgoing Messages 119
4.5.6. Counter Processing of Incoming Messages 120
4.6. Message Processing 121
4.6.1. Message Transmission 121
4.6.2. Message Reception 121
4.7. Message Security 122
4.7.1. Data confidentiality and integrity with data origin authentication parameters 122
4.7.2. Security Processing of Outgoing Messages 123
4.7.3. Security Processing of Incoming Messages 126
4.8. Message Privacy 127
4.8.1. Privacy Key 127
4.8.2. Privacy Nonce 127
4.8.3. Privacy Processing of Outgoing Messages 128
4.8.4. Privacy Processing of Incoming Messages 129
4.9. Message Exchanges 129
4.9.1. Exchange Role 129
4.9.2. Exchange ID 130
4.9.3. Exchange Context 130
4.9.4. Exchange Message Dispatch 130
4.9.5. Exchange Message Processing 131
4.10. Secure Channel Protocol 133
4.10.1. Secure Channel Protocol Messages 133
4.10.2. Parameters and Constants 136
4.11. Message Reliability Protocol (MRP) 136
4.11.1. Reliable Messaging Header Fields 137
4.11.2. Reliable transfer 137

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

4.11.3. Peer Exchange Management 139
4.11.4. Transport Considerations 140
4.11.5. Reliable Message Processing 140
4.11.6. Reliable Message State 145
4.11.7. MRP Messages 145
4.11.8. Parameters and Constants 146
4.12. Unicast Communication 146
4.12.1. Session Establishment Phase 147
4.12.2. Application Data Phase 149
4.13. Session Establishment 150
4.13.1. Passcode-Authenticated Session Establishment (PASE) 150
4.13.2. Certificate Authenticated Session Establishment (CASE) 157
4.14. Group Communication 179
4.14.1. Groupcast Session Context 179
4.14.2. Sending a group message 180
4.14.3. Receiving a group message 180
4.15. Group Key Management 181
4.15.1. Operational Groups 181
4.15.2. Operational Group Key Derivation 182
4.15.3. Epoch Keys 183
4.15.4. Distribution of Key Material 187
4.16. Message Counter Synchronization Protocol (MCSP) 190
4.16.1. Message Counter Synchronization Methods 190
4.16.2. Group Peer State 191
4.16.3. MCSP Messages 191
4.16.4. Unsynchronized Message Processing 192
4.16.5. Message Counter Synchronization Exchange 193
4.16.6. Message Counter Synchronization Session Context 195
4.16.7. Sequence Diagram 196
4.17. Bluetooth Transport Protocol (BTP) 198
4.17.1. BTP Session Interface 198
4.17.2. BTP Frame Formats 199
4.17.3. BTP GATT Service 202
4.17.4. Parameters and Constants 213
4.17.5. Bluetooth SIG Considerations 214
5. Commissioning 215
5.1. Onboarding Payload 215
5.1.1. Onboarding Payload Contents 215
5.1.2. Onboarding Material Representation 216
5.1.3. QR Code 217
5.1.4. Manual Pairing Code 221

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

5.1.5. TLV Content 224
5.1.6. Concatenation 227
5.1.7. Generation of the Passcode 228
5.1.8. NFC Tag 229
5.2. Initiating Commissioning 229
5.2.1. Purpose and Scope 230
5.2.2. User Journey Details 230
5.3. User Directed Commissioning 235
5.3.1. Overview 236
5.3.2. UDC Protocol Messages 236
5.3.3. Message format 237
5.3.4. Message Exchanges 237
5.3.5. IdentificationDeclaration Message 237
5.4. Device Discovery 238
5.4.1. Purpose and Scope 238
5.4.2. Announcement by Device 238
5.4.3. Discovery by Commissioner 249
5.5. Commissioning Flows 250
5.5.1. Commissioning Flows Error Handling 253
5.5.2. Commissioning Flow Diagrams 255
5.6. Administrator Assisted Commissioning Flows 256
5.6.1. Introduction 256
5.6.2. Basic Commissioning Method (BCM) 257
5.6.3. Enhanced Commissioning Method (ECM) 257
5.6.4. Open Commissioning Window 259
5.7. Device Commissioning Flows 259
5.7.1. Standard Commissioning Flow 259
5.7.2. User-Intent Commissioning Flow 260
5.7.3. Custom Commissioning Flow 261
5.7.4. Manual Pairing Code and QR Code Inclusion 268
5.8. In-field Upgrade to Matter 270
6. Device Attestation and Operational Credentials 271
6.1. Common Conventions 271
6.1.1. Encoding of Matter-specific RDNs 271
6.1.2. Key Identifier Extension Constraints 273
6.1.3. Certificate Sizes 273
6.1.4. Presentation of example certificates 273
6.2. Device Attestation 274
6.2.1. Introduction 274
6.2.2. Device Attestation Certificate (DAC) 274
6.2.3. Device Attestation Procedure 287

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

6.3. Certification Declaration 290
6.3.1. Certification Declaration (CD) Format 290
6.3.2. Firmware Information 293
6.3.3. Firmware information validation examples 295
6.4. Node Operational Credentials Specification 297
6.4.1. Introduction 297
6.4.2. Node Operational Credentials Management 297
6.4.3. Node Operational Identifier Composition 298
6.4.4. Node Operational Key Pair 299
6.4.5. Node Operational Credentials Certificates 299
6.4.6. Node Operational Credentials Procedure 300
6.4.7. Node Operational Certificate Signing Request (NOCSR) 302
6.4.8. Node Operational Certificate Renewal 303
6.4.9. Node Operational Certificate Revocation 303
6.4.10. Security Considerations 303
6.5. Operational Certificate Encoding 303
6.5.1. Introduction 303
6.5.2. Matter certificate 304
6.5.3. Version Number 305
6.5.4. Serial Number 305
6.5.5. Signature Algorithm 305
6.5.6. Issuer and Subject 306
6.5.7. Validity 311
6.5.8. Public Key Algorithm 311
6.5.9. EC Curve Identifier 311
6.5.10. Public Key 312
6.5.11. Extensions 312
6.5.12. Matter certificate Extensions Encoding Rules 316
6.5.13. Signature 317
6.5.14. Invalid Matter certificates 317
6.5.15. Examples 318
6.6. Access Control 325
6.6.1. Scope and Purpose 325
6.6.2. Model 325
6.6.3. Access Control List Examples 329
6.6.4. Access Control Cluster update side-effects 334
6.6.5. Conceptual Access Control Privilege Granting Algorithm 335
6.6.6. Applying Privileges to Action Paths 340
7. Data Model Specification 341
7.1. Practical Information 341
7.1.1. Revision History 341

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

7.1.2. Scope & Purpose 341
7.1.3. Origin Story 341
7.1.4. Overview 341
7.1.5. Glossary 342
7.1.6. Conventions 342
7.1.7. Reserved Bit Fields 342
7.2. Data Qualities 343
7.2.1. Common Data Table Columns 343
7.2.2. Other Data Table Columns 344
7.3. Conformance 344
7.3.1. Optional 345
7.3.2. Provisional 345
7.3.3. Mandatory 346
7.3.4. Disallowed 346
7.3.5. Deprecated 346
7.3.6. Exclusivity 346
7.3.7. List 346
7.3.8. Expressions and Optionality 347
7.3.9. Choice 348
7.3.10. Blank Conformance 349
7.4. Element 349
7.4.1. Encoded Element Processing 350
7.5. Fabric 350
7.5.1. Accessing Fabric 350
7.5.2. Fabric-Index 350
7.5.3. Fabric-Scoped Data 351
7.5.4. Fabric-Scoped IDs 351
7.6. Access 352
7.6.1. Read Access 353
7.6.2. Write Access 353
7.6.3. Invoke Access 353
7.6.4. Fabric-Scoped Quality 354
7.6.5. Fabric-Sensitive Quality 354
7.6.6. View Privilege 354
7.6.7. Operate Privilege 354
7.6.8. Manage Privilege 354
7.6.9. Administer Privilege 355
7.6.10. Timed Interaction 355
7.7. Other Qualities 355
7.7.1. Nullable Quality 356
7.7.2. Non-Volatile Quality 356

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

7.7.3. Fixed Quality . . . 356
7.7.4. Scene Quality . . . 356
7.7.5. Reportable Quality . . . 357
7.7.6. Changes Omitted Quality . . . 357
7.7.7. Singleton . . . 357
7.8. Node . . . 357
7.9. Endpoint . . . 357
7.10. Cluster . . . 358
7.10.1. Cluster Revision . . . 358
7.10.2. Cluster Optional Features . . . 359
7.10.3. Cluster Data Version . . . 359
7.10.4. New Cluster . . . 360
7.10.5. Cluster Aliasing . . . 360
7.10.6. Cluster Inheritance . . . 360
7.10.7. Status Codes . . . 361
7.10.8. Cluster Classification . . . 362
7.11. Command . . . 363
7.11.1. Command Fields . . . 364
7.12. Attribute . . . 365
7.12.1. Persistence . . . 365
7.13. Global Elements . . . 366
7.13.1. ClusterRevision Attribute . . . 367
7.13.2. FeatureMap Attribute . . . 367
7.13.3. AttributeList Attribute . . . 368
7.13.4. AcceptedCommandList Attribute . . . 368
7.13.5. GeneratedCommandList Attribute . . . 368
7.13.6. EventList Attribute . . . 368
7.13.7. FabricIndex Field . . . 369
7.14. Event . . . 369
7.14.1. Priority . . . 369
7.14.2. Event Record . . . 369
7.14.3. Buffering . . . 370
7.14.4. Event Filtering . . . 370
7.14.5. Fabric-Sensitive Event . . . 371
7.15. Device Type . . . 371
7.15.1. Device Type Revision . . . 372
7.15.2. Device Type Composition . . . 372
7.15.3. Device Type Classification . . . 372
7.15.4. Extra Clusters on an Endpoint . . . 373
7.16. Non-Standard . . . 374
7.17. Data Field . . . 374

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

7.17.1. Nullable 375
7.17.2. Optional or Deprecated 375
7.17.3. Constraint & Value 375
7.17.4. Default Column 378
7.18. Data Types 379
7.18.1. Base Data Types 379
7.18.2. Derived Data Types 388
7.19. Manufacturer Specific Extensions 397
7.19.1. Manufacturer Extensible Identifiers 397
7.19.2. Manufacturer Extensible Identifier (MEI) 398
7.19.3. Manufacturer Extensions 400
7.19.4. Discoverability 403
8. Interaction Model Specification 405
8.1. Practical Information 405
8.1.1. Revision History 405
8.1.2. Scope & Purpose 405
8.1.3. Origin Story 405
8.1.4. Purpose 406
8.1.5. Glossary 406
8.1.6. Conventions & Conformance 407
8.2. Concepts 407
8.2.1. Path 407
8.2.2. Interaction 410
8.2.3. Transaction 411
8.2.4. Action 411
8.2.5. Common Action Behavior 412
8.3. Status and Interaction 414
8.3.1. Status Response Action 414
8.4. Read Interaction 415
8.4.1. Read Transaction 416
8.4.2. Read Request Action 416
8.4.3. Report Data Action 417
8.5. Subscribe Interaction 420
8.5.1. Subscribe Transaction 422
8.5.2. Subscribe Request Action 422
8.5.3. Subscribe Response Action 423
8.6. Report Transaction 424
8.6.1. Report Transaction Non-Empty 425
8.6.2. Report Transaction Empty 425
8.7. Write Interaction 425
8.7.1. Write Transaction 425

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

8.7.2. Write Request Action 426
8.7.3. Write Response Action 427
8.7.4. Timed Request Action 429
8.8. Invoke Interaction 429
8.8.1. Invoke Transaction 429
8.8.2. Invoke Request Action 430
8.8.3. Invoke Response Action 433
8.9. Common Action Information Blocks and Paths 434
8.9.1. Path Information 434
8.9.2. Attribute Information Blocks 434
8.9.3. Event Information Blocks and Paths 440
8.9.4. Command Information Blocks and Paths 442
8.9.5. Status Information Blocks and Paths 443
8.10. Status Codes 444
8.10.1. Status Code Table 445
9. System Model Specification 449
9.1. Practical Information 449
9.1.1. Revision History 449
9.1.2. Scope and Purpose 449
9.1.3. Origin Story 449
9.1.4. Overview 449
9.2. Endpoint Composition 449
9.2.1. Dynamic Endpoint allocation 451
9.3. Interaction Model Relationships 452
9.3.1. Subscription 452
9.4. Binding Relationship 452
9.5. Descriptor Cluster 453
9.5.1. Revision History 453
9.5.2. Classification 453
9.5.3. Cluster Identifiers 454
9.5.4. Attributes 454
9.5.5. Data Types 455
9.6. Binding Cluster 455
9.6.1. Binding Mutation 456
9.6.2. Revision History 456
9.6.3. Classification 456
9.6.4. Cluster Identifiers 456
9.6.5. Attributes 457
9.6.6. Data Types 457
9.7. Label Cluster 458
9.7.1. Revision History 458

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

9.7.2. Classification 458
9.7.3. Cluster Identifiers 458
9.7.4. Attributes 458
9.7.5. Data Types 459
9.8. Fixed Label Cluster 459
9.8.1. Revision History 460
9.8.2. Classification 460
9.8.3. Cluster Identifiers 460
9.8.4. Attributes 460
9.9. User Label Cluster 460
9.9.1. Revision History 460
9.9.2. Classification 460
9.9.3. Cluster Identifiers 460
9.9.4. Attributes 461
9.10. Access Control Cluster 461
9.10.1. Revision History 461
9.10.2. Classification 461
9.10.3. Cluster Identifiers 461
9.10.4. Features 462
9.10.5. Attributes 462
9.10.6. Error handling 471
9.10.7. Events 472
9.10.8. Data Types 474
9.11. Group Relationship 474
9.12. Bridge for non-Matter devices 475
9.12.1. Introduction 475
9.12.2. Exposing functionality and metadata of Bridged Devices 476
9.12.3. Discovery of Bridged Devices 480
9.12.4. Configuration of Bridged Devices 480
9.12.5. New features for Bridged Devices 482
9.12.6. Changes to the set of Bridged Devices 483
9.12.7. Changes to device names and grouping of Bridged Devices 483
9.12.8. Setup flow for a Bridge (plus Bridged Devices) 483
9.12.9. Access Control 483
9.12.10. Software update (OTA) 484
9.12.11. Best practices for Bridge Manufacturers 484
9.12.12. Best practices for Administrators 485
9.13. Bridged Device Basic Information Cluster 485
9.13.1. Scope & Purpose 485
9.13.2. Revision History 486
9.13.3. Classification 486

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

9.13.4. Cluster Identifiers 486
9.13.5. Features 486
9.13.6. Attributes 486
9.13.7. Events 487
9.14. Actions Cluster 488
9.14.1. Scope & Purpose 488
9.14.2. Revision History 489
9.14.3. Classification 489
9.14.4. Cluster Identifiers 489
9.14.5. Features 489
9.14.6. Attributes 489
9.14.7. Commands 490
9.14.8. Events 496
9.14.9. Data Types 497
9.14.10. Examples 502
9.15. Proxy Architecture 508
9.15.1. Motivation 508
9.15.2. Subscription Proxy: Overview 508
9.15.3. Composition & Paths 509
9.15.4. Proxy Subscriptions 510
9.15.5. Schemas and Data Serialization/Deserialization 512
9.15.6. Indirect Proxies 512
9.15.7. Proxy Discovery & Assignment Flow 512
9.15.8. Constraints 519
9.15.9. Certification 520
9.15.10. Security & Privacy 520
9.15.11. Parameters and Constants 521
9.15.12. Clusters 521
9.15.13. Proxy Discovery Cluster 521
9.15.14. Proxy Configuration Cluster 524
9.15.15. Valid Proxies Cluster 525
10. Interaction Model Encoding Specification 529
10.1. Overview 529
10.2. Messages 529
10.2.1. IM Protocol Messages 529
10.2.2. Common Action Information Encoding 529
10.2.3. Chunking 530
10.2.4. Transaction Flows 531
10.3. Data Types 534
10.3.1. Analog - Integer 535
10.3.2. Analog - Floating Point 535

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

10.3.3. Discrete - Enumeration 535
10.3.4. Discrete - Bitmap 536
10.3.5. Composite - String 536
10.3.6. Composite - Octet String 536
10.3.7. Collection - Struct 536
10.3.8. Collection - List 536
10.3.9. Derived Types 536
10.3.10. Field IDs 536
10.4. Sample Cluster 536
10.4.1. Disco Ball Cluster 536
10.4.2. Super Disco Ball Cluster 544
10.5. Information Blocks 545
10.5.1. Tag Rules 546
10.5.2. AttributePathIB 546
10.5.3. DataVersionFilterIB 549
10.5.4. AttributeDataIB 549
10.5.5. AttributeReportIB 552
10.5.6. EventFilterIB 553
10.5.7. ClusterPathIB 553
10.5.8. EventPathIB 553
10.5.9. EventDataIB 554
10.5.10. EventReportIB 555
10.5.11. CommandPathIB 555
10.5.12. CommandDataIB 556
10.5.13. InvokeResponseIB 557
10.5.14. CommandStatusIB 557
10.5.15. EventStatusIB 558
10.5.16. AttributeStatusIB 558
10.5.17. StatusIB 558
10.6. Message Definitions 558
10.6.1. StatusResponseMessage 558
10.6.2. ReadRequestMessage 558
10.6.3. ReportDataMessage 559
10.6.4. SubscribeRequestMessage 562
10.6.5. SubscribeResponseMessage 562
10.6.6. WriteRequestMessage 562
10.6.7. WriteResponseMessage 563
10.6.8. TimedRequestMessage 563
10.6.9. InvokeRequestMessage 563
10.6.10. InvokeResponseMessage 564
11. Service and Device Management 565

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

11.1. Basic Information Cluster 565
11.1.1. Scope & Purpose 565
11.1.2. Revision History 565
11.1.3. Classification 565
11.1.4. Cluster Identifiers 565
11.1.5. Features 565
11.1.6. Server 565
11.2. Group Key Management Cluster 572
11.2.1. Scope & Purpose 572
11.2.2. Revision History 572
11.2.3. Classification 572
11.2.4. Cluster Identifiers 572
11.2.5. Features 573
11.2.6. Data Types 573
11.2.7. Server 576
11.2.8. Client 577
11.2.9. Commands 577
11.3. Localization Configuration Cluster 580
11.3.1. Scope & Purpose 580
11.4. Time Format Localization Cluster 581
11.4.1. Scope & Purpose 581
11.4.2. Features 582
11.4.3. Data Types 582
11.4.4. Attributes 583
11.5. Unit Localization Cluster 584
11.5.1. Scope & Purpose 584
11.5.2. Features 585
11.5.3. Data Types 585
11.5.4. Attributes 585
11.6. Power Source Configuration Cluster 586
11.6.1. Revision History 586
11.6.2. Classification 586
11.6.3. Cluster Identifiers 586
11.6.4. Features 586
11.6.5. Server 586
11.6.6. Client 587
11.6.7. Commands 587
11.7. Power Source Cluster 587
11.7.1. Revision History 587
11.7.2. Classification 587
11.7.3. Cluster Identifiers 588

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

11.7.4. Features 588
11.7.5. Data Types 588
11.7.6. Server 590
11.7.7. Client 603
11.7.8. Commands 603
11.7.9. Configuration Examples 603
11.8. Network Commissioning Cluster 606
11.8.1. Scope & Purpose 606
11.8.2. Revision History 607
11.8.3. Classification 607
11.8.4. Cluster Identifiers 607
11.8.5. Features 607
11.8.6. Data Types 607
11.8.7. Attributes 611
11.8.8. Commands 613
11.8.9. Usage of networking configurations 625
11.9. General Commissioning Cluster 627
11.9.1. Revision History 628
11.9.2. Classification 628
11.9.3. Cluster Identifiers 628
11.9.4. Features 628
11.9.5. Data Types 628
11.9.6. Server Attributes 630
11.9.7. Commands 631
11.10. Diagnostic Logs Cluster 637
11.10.1. Scope & Purpose 637
11.10.2. Revision History 638
11.10.3. Classification 638
11.10.4. Cluster Identifiers 638
11.10.5. Features 638
11.10.6. Data Types 638
11.10.7. Server 640
11.10.8. Client 640
11.10.9. Commands 640
11.11. General Diagnostics Cluster 642
11.11.1. Scope & Purpose 643
11.11.2. Revision History 643
11.11.3. Classification 643
11.11.4. Cluster Identifiers 643
11.11.5. Features 643
11.11.6. Data Types 643

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

11.11.7. Attributes 648
11.11.8. Commands 651
11.11.9. Events 652
11.11.10. Status Codes 653
11.12. Software Diagnostics Cluster 654
11.12.1. Scope & Purpose 654
11.12.2. Revision History 654
11.12.3. Classification 654
11.12.4. Cluster Identifiers 654
11.12.5. Features 654
11.12.6. Data Types 654
11.12.7. Attributes 655
11.12.8. Commands 656
11.12.9. Events 657
11.13. Thread Network Diagnostics Cluster 657
11.13.1. Scope & Purpose 658
11.13.2. Revision History 658
11.13.3. Classification 658
11.13.4. Cluster Identifiers 658
11.13.5. Features 658
11.13.6. Data Types 659
11.13.7. Attributes 659
11.13.8. Commands 677
11.13.9. Events 678
11.14. Wi-Fi Network Diagnostics Cluster 679
11.14.1. Scope & Purpose 679
11.14.2. Features 679
11.14.3. Data Types 680
11.14.4. Attributes 681
11.14.5. Commands 683
11.14.6. Events 684
11.15. Ethernet Network Diagnostics Cluster 686
11.15.1. Scope & Purpose 686
11.15.2. Features 686
11.15.3. Data Types 687
11.15.4. Attributes 687
11.15.5. Events 689
11.15.6. Commands 689
11.16. Time Synchronization 689
11.16.1. Revision History 690
11.16.2. Classification 690

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

11.16.3. Cluster Identifiers 690
11.16.4. Terminology 690
11.16.5. Features 690
11.16.6. Attributes 691
11.16.7. Commands 694
11.16.8. Events 695
11.16.9. Data Types 696
11.16.10. Time Synchronization at Commissioning 700
11.16.11. Time Synchronization during operation 700
11.16.12. Time source prioritization 701
11.16.13. Time synchronization maintenance 701
11.16.14. Acting as an NTP Server 701
11.16.15. Implementation Guidance 702
11.17. Node Operational Credentials Cluster 704
11.17.1. Revision History 704
11.17.2. Classification 705
11.17.3. Cluster Identifiers 705
11.17.4. Features 705
11.17.5. Data Types 705
11.17.6. Attributes 711
11.17.7. Commands 713
11.18. Administrator Commissioning Cluster 725
11.18.1. Administrator Commissioning Cluster 725
11.18.2. Revision History 725
11.18.3. Classification 726
11.18.4. Cluster Identifiers 726
11.18.5. Features 726
11.18.6. Data Types 726
11.18.7. Attributes 727
11.18.8. Commands 728
11.18.9. Status Codes 731
11.19. Over-the-Air (OTA) Software Update 731
11.19.1. Scope & Purpose 731
11.19.2. Functional overview 732
11.19.3. Software update workflow 733
11.19.4. Security considerations 750
11.19.5. Some special situations 752
11.19.6. OTA Software Update Provider Cluster Definition 753
11.19.7. OTA Software Update Requestor Cluster Definition 762
11.20. Over-the-Air (OTA) Software Update File Format 771
11.20.1. Scope & Purpose 771

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

11.20.2. General Structure 772
11.20.3. Security considerations 775
11.21. Bulk Data Exchange Protocol (BDX) 775
11.21.1. Overview 775
11.21.2. Terminology 776
11.21.3. Protocol Opcodes and Status Report Values 777
11.21.4. Security and Transport Constraints 779
11.21.5. Transfer Management Messages 779
11.21.6. Data Transfer Messages 789
11.21.7. Synchronous Transfers Message Flows 793
11.21.8. Asynchronous Tranfers Message Flows 802
11.22. Distributed Compliance Ledger 804
11.22.1. Scope & Purpose 804
11.22.2. Schemas 805
11.22.3. Vendor Schema 806
11.22.4. PAA Schema 807
11.22.5. DeviceModel Schema 807
11.22.6. DeviceSoftwareVersionModel Schema 811
11.22.7. DeviceSoftwareCompliance / Compliance test result Schema 814
11.22.8. APIs / CLI 815
12. Multiple Fabrics 817
12.1. Multiple Fabrics 817
12.1.1. Introduction 817
12.1.2. User Consent 817
12.1.3. Administrator-Assisted Commissioning Method 817
12.1.4. Node Behavior 817
13. Security Requirements 819
13.1. Overview 819
13.2. Device vs. Node 819
13.3. Commissioning 819
13.4. Factory Reset 820
13.5. Firmware 820
13.6. Security Best Practices 820
13.6.1. Cryptography 821
13.6.2. Commissioning 821
13.6.3. Firmware 821
13.6.4. Manufacturing 822
13.6.5. Resiliency 822
13.6.6. Battery Powered Devices 822
13.6.7. Tamper Resistance 822
13.6.8. Bridging 822

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

13.6.9. Distributed Compliance Ledger . . . 822
13.7. Threats and Countermeasures . . . 823
Appendix A: Tag-length-value (TLV) Encoding Format . . . 835
A.1. Scope & Purpose . . . 835
A.2. Tags . . . 835
A.2.1. Profile-Specific Tags . . . 835
A.2.2. Context-Specific Tags . . . 835
A.2.3. Anonymous Tags . . . 836
A.2.4. Canonical Ordering of Tags . . . 836
A.3. Lengths . . . 836
A.4. Primitive Types . . . 836
A.5. Container Types . . . 837
A.5.1. Structures . . . 837
A.5.2. Arrays . . . 837
A.5.3. Lists . . . 837
A.6. Element Encoding . . . 838
A.7. Control Octet Encoding . . . 838
A.7.1. Element Type Field . . . 838
A.7.2. Tag Control Field . . . 840
A.8. Tag Encoding . . . 840
A.8.1. Fully-Qualified Tag Form . . . 840
A.8.2. Implicit Profile Tag Form . . . 841
A.8.3. Common Profile Tag Form . . . 841
A.8.4. Context-Specific Tag Form . . . 841
A.8.5. Anonymous Tag Form . . . 841
A.9. Length Encoding . . . 841
A.10. End of Container Encoding . . . 842
A.11. Value Encodings . . . 842
A.11.1. Integers . . . 842
A.11.2. UTF-8 and Octet Strings . . . 842
A.11.3. Booleans . . . 842
A.11.4. Arrays, Structures and Lists . . . 843
A.11.5. Floating Point Numbers . . . 843
A.11.6. Nulls . . . 843
A.12. TLV Encoding Examples . . . 843
Appendix B: Tag-length-value (TLV) Schema Definitions . . . 847
B.1. Introduction . . . 847
B.1.1. Basic Structure . . . 847
B.1.2. Keywords . . . 847
B.1.3. Naming . . . 847
B.1.4. Namespaces . . . 848

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

B.1.5. Qualifiers 848
B.1.6. Tagging 849
B.2. Definitions 849
B.2.1. Type Definition (type-def) 849
B.2.2. FIELD GROUP Definition (field-group-def) 850
B.2.3. Namespace Definition (namespace-def) 851
B.2.4. PROTOCOL Definition (protocol-def) 853
B.2.5. VENDOR Definition (vendor-def) 854
B.3. Types 854
B.3.1. ARRAY / ARRAY OF 854
B.3.2. BOOLEAN 856
B.3.3. FLOAT32 / FLOAT64 857
B.3.4. SIGNED INTEGER / UNSIGNED INTEGER 857
B.3.5. LIST / LIST OF 858
B.3.6. OCTET STRING 859
B.3.7. NULL 860
B.3.8. STRING 860
B.3.9. STRUCTURE 860
B.4. Pseudo-Types 863
B.4.1. ANY 863
B.4.2. CHOICE OF 863
B.5. Qualifiers 866
B.5.1. any-order / schema-order / tag-order 866
B.5.2. extensible 866
B.5.3. id 867
B.5.4. length 868
B.5.5. nullable 868
B.5.6. optional 869
B.5.7. range 869
B.5.8. tag 870
B.5.9. Documentation and Comments 872
Appendix C: Tag-length-value (TLV) Payload Text Representation Format 873
C.1. Introduction 873
C.2. Format Specification 873
C.2.1. Tag/Value 873
C.2.2. Context-Specific Tags 873
C.2.3. Protocol-Specific Tags 873
C.2.4. Anonymous Tags 874
C.2.5. Primitive Types 874
C.2.6. Complex Types: Structure 875
C.2.7. Complex Types: Arrays 875

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

C.2.8. Complex Types: List . . . 875
C.3. Examples . . . 875
C.3.1. TLV Schema . . . 875
C.3.2. TLV Payloads . . . 876
Appendix D: Status Report Messages . . . 879
D.1. Overview . . . 879
D.2. Status Report elements . . . 879
D.3. Message Format . . . 879
D.3.1. General status codes (GeneralCode) . . . 880
D.3.2. Protocol-specific codes (ProtocolId and ProtocolCode) . . . 880
D.3.3. Protocol-specific data (ProtocolData) . . . 881
D.4. Presenting StatusReport messages in protocol specifications . . . 881
Appendix E: Matter-Specific ASN.1 Object Identifiers (OIDs) . . . 883
Appendix F: Cryptographic test vectors for some procedures . . . 885
F.1. Certification Declaration CMS test vector . . . 885
F.2. Device Attestation Response test vector . . . 888
F.3. Node Operational CSR Response test vector . . . 891
Appendix G: Minimal Resource Requirements . . . 895

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Chapter 1. Introduction

The Matter specification defines fundamental requirements to enable an interoperable application layer solution for smart home devices over the Internet Protocol.

1.1. Scope and Purpose

This specification details everything necessary to implement an application and transport layer stack. It is intended to be used by implementers as a complete specification but where necessary other references are noted with details on how these references apply to this specification.

In case of discrepancies between this specification and the SDK [https://github.com/project-chip/connectedhomeip/], this specification SHALL take precedence.

1.2. Acronyms and Abbreviations

Acronym	Definition
ACL	Access Control List
AGID	Application Group Identifier
AEAD	Authenticated Encryption with Associated Data
AES	Advanced Encryption Standard (from FIPS 197)
AP	Access Point
API	Application Programming Interface
ASN.1	Abstract Syntax Notation 1 (from ITU-T X.680)
BLE	Bluetooth Low Energy
BDX	Bulk Data Exchange
BTP	Bluetooth Transport Protocol
CA	Certificate Authority (also known as Certification Authority)
CASE	Certificate Authenticated Session Establishment
CAT	CASE Authenticated Tag
CBC-MAC	Cipher Block Chaining Message Authentication Code
CCM	Counter mode of encryption with CBC-MAC (AEAD mode) (from NIST SP 800-38C)
CD	Certification Declaration
CMS	Cryptographic Message Syntax
CN	Common Name (from X.520)
CSR	Certificate Signing Request
CTR	Counter Mode (AES block cipher mode) (from NIST SP 800-38A)

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Acronym	Definition
DAC	Device Attestation Certificate
DER	Distinguished Encoding Rule (from X.690)
DN	Distinguished Name (from X.520)
DNS	Domain Name System
DNS-SD	DNS Based Service Discovery (from RFC 6763)
DRBG	Deterministic Random Bit Generator (from NIST 800-90A)
ECC	Elliptic Curve Cryptography (from SEC 1) (also "Error Correction Code")
ECDHE	Elliptic Curve Ephemeral Diffie-Hellman (from SEC 1)
ECDSA	Elliptic Curve Digital Signature Algorithm (from SEC 1)
EUI	Extended Unique Identifier
EUI-64	64-bit EUI
GATT	Bluetooth Generic Attribute Profile
GID	Group Identifier (also referred to as "Group ID")
GKH	Group Key Hash
GUA	Global Unicast Address
HMAC	Keyed-Hash Message Authentication Code (from FIPS 198-1)
ID	Identifier
IP	Internet Protocol
IPK	Identity Protection Key (a Universal Group key shared across a Fabric)
KDF	Key Derivation Function (from NIST 800-56C)
KDM	Key Derivation Method (from NIST 800-56C)
LLA	Link local address
LLN	Low power and Lossy Network
MAC	Medium Access Control (or "Message Authentication Code")
MCSP	Message Counter Synchronization Protocol
MIC	Message Integrity Code (used as synonym for MAC (Message Authentication Code) to avoid confusion with MAC (Medium Access Control) as used in network addressing contexts)
MRP	Message Reliability Protocol
NFC	Near Field Communication
NOC	Node Operational Certificate
NOCSR	Node Operational Certificate Signing Request

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Acronym	Definition
OID	Object Identifier (from ITUÊASN.1)
OTA	Over-the-air (used mostly in context of "Over-the-air Software Update")
PAA	Product Attestation Authority
PAI	Product Attestation Intermediate
PAKE	Password-Authenticated Key Exchange (from SPAKE2+)
PASE	Passcode-Authenticated Session Establishment
PBKDF	Password-Based Key Derivation Function (from NISTÊ800-132)
PDU	Protocol Data Unit
PKI	Public Key Infrastructure
PID	Product Identifier (also Product ID)
PIN	Personal Identification Number
QR code	Quick Response (code)
SDU	Service Data Unit
SED	Sleepy End Device
SHA	Secure Hash Algorithm (from FIPSÊ180-4)
SRP	Service Registration Protocol (from SRP)
TCP	Transmission Control Protocol
TFTP	Trivial File Transfer Protocol (from RFCÊ1350)
TLV	Tag Length Value (refers mostly to Tag-length-value (TLV) Encoding Format)
TRNG	True Random Number Generator (from NISTÊ800-90B)
UDP	User Datagram Protocol
UGID	Universal Group Identifier
ULA	Unique local address
UTC	Universal Time Coordinated
UUID	Universally Unique Identifier
VID	Vendor Identifier (also Vendor ID)
ZCL	Zigbee Cluster Library

1.3. Definitions

Term	Definition
Access Control List	A list of entries in the Access Control Cluster expressing individual rules which grant privileges to access cluster elements.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Term	Definition
Administrator	A Node having Administer privilege over at least the Access Control Cluster of another Node.
Advertising Data	A data container used in BLE Advertisements to convey a logical grouping of information.
Attribute	A data entity which represents a physical quantity or state. This data is com! municated to other Nodes using commands.
Binding	A persistent attachment between an instance on one Node to one-or-more cor! responding instances on another (or the same) Node.
Border Router	A router, also known as Edge Router, that provides routing services between two IP subnets (typically, between a hub network and a peripheral network).
Bridge	A Node that represents one or more non-Matter devices on the Fabric.
Bridged Device	A non-Matter device that is represented on the Fabric by a Bridge so it can be used by Nodes on the Fabric.
Broadcast	The transmission of a message to every Node in a particular broadcast domain, be it all Nodes on a Ethernet or Wi-Fi link, and/or all Nodes on a Thread mesh.
Certificate Author! ity (CA)	An entity that issues digital certificates such as a DAC or NOC
Certification Dec! laration	A digitally signed token that conveys Matter certification status of a vendorÕs certified Device.
Client	A Cluster interface that typically sends commands that manipulate the attributes on the corresponding server cluster. A client cluster communicates with a corresponding remote server cluster with the same cluster identifier.
Cluster	A specification defining one or more attributes, commands, behaviors and dependencies, that supports an independent utility or application function. The term may also be used for an implementation or instance of such a specifi! cation on an endpoint.
Command	Requests for action on a value with an expected response which may have parameters and a response with a status and parameters.
Commission	To bring a Node into a Fabric.
Commissionable Node	A Node that is able to be commissioned. Specific actions such as a button press may be required to put a Commissionable Node into Commissioning Mode in order for it to allow Commissioning.
Commissionable Node Discovery	Discovery of a Node that is able to be Commissioned, but not necessarily in Commissioning Mode, for the purpose of performing Commissioning. The Node may be brand new, after factory reset, or it may have have already been Commissioned.
Commissioner	A Role of a Node that performs Commissioning.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Term	Definition
Commissioner Dis! covery	Discovery of a Commissioner.
Commissionee	An entity that is being Commissioned to become a Node.
Commissioning	Sequence of operations to bring a Node into a Fabric by assigning an Opera! tional Node ID and Node Operational credentials.
Commissioning Channel	A Secure Channel used to perform Commissioning.
Commissioning Mode	The mode of a Node in which it allows Commissioning.
Controller	A Role of a Node that has permissions to enable it to control one or more Nodes.
Controlee	A Role of a Node that has permissions defined to enable it to be controlled by one or more Nodes.
Device	A piece of equipment containing one or more Nodes.
Device Attestation Certificate	An RFCÊ5280[https://www.rfc-editor.org/rfc/rfc5280] compliant X.509 v3 document with attestable attributes.
Discriminator	A 12-bit value used to discern between multiple commissionable Matter device advertisements. See Discriminator value .
Endpoint	A particular component within a Node that is individually addressable.
Endpoint Address	The address assigned to an Endpoint.
Fabric	A logical collection of communicating Nodes, sharing a common root of trust, and a common distributed configuration state.
Information Ele! ment	A Wi-Fi (IEEEÊ802.11-2020) data container used to convey various information regarding a particular Wi-Fi networkÕs capabilities and operation.
Key Center	A system component which takes the NOCSR from a Commissioner and allo! cates an Operational Node ID that is unique to the Fabric, inserts this Opera! tional Node ID as the DN into the NOC, and signs the NOC.
Manual Pairing Code	An 11-digit or 21-digit numeric code that can be manually entered/spoken instead of scanning a QR code, which contains the information needed to com! mission a Matter device.
Network	A set of nodes that have addressability, connectivity, and reachability to one another via Internet Protocol.
Node	An addressable entity which supports the Matter protocol stack and (once Commissioned) has its own Operational Node ID and Node Operational cre! dentials. A Device MAY host multiple Nodes.
Operational Dis! covery	Discovery of a previously commissioned Node for the purpose of performing operations with that Node.
Onboarding Pay! load	The information needed to start the process of commissioning a Device.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Term	Definition
OTA Provider	A Node implementing the OTA Software Update Provider role (see OTA Software Update Provider Cluster Definition).
OTA Requestor	A Node implementing the OTA Software Update Requestor role (see OTA Software Update Requestor Cluster Definition).
Product Attestation Authority	An entity which operates a root level Certificate Authority for the purpose of Device Attestation.
Product Attestation Intermediate	An entity which operates an intermediate level Certificate Authority for the purpose of Device Attestation.
Product ID (PID)	A 16-bit number that identifies the type of a Device, uniquely among the product types made by a given vendor. See Product ID .
QR Code	A machine-readable optical label that contains information about the item to which it is attached (see QR Code).
Role	Some set of (related) behaviors of a Node. Each Node can have multiple roles.
Router	A device that provides routing services in its network in cooperation with other Routers.
Soft-AP	A device utilizing Wi-Fi (IEEE 802.11-2020) Access Point (AP) functionality to advertise its presence and allow IP-bearing connections but does not offer Internet connectivity.
Secure Channel	A channel in which messages are encrypted and authenticated. Unicast secure channels also provide authentication of each peer.
Server	A Cluster interface that typically supports all or most of the attributes of the Cluster. A Server Cluster communicates with a corresponding remote Client Cluster with the same Cluster identifier.
Service Discovery	The ability of a Node to locate services of interest.
Software Image	A data blob, equivalent to a file, utilized by a Node to update its software. For the purposes of OTA Software Update, this further refers to files conforming to the OTA Software Image File Format .
Thread	A low-power IEEE 802.15.4-based IPv6 mesh networking technology (see Thread specification).
Vendor	The organization that made a Device.
Vendor ID (VID)	A 16-bit number that uniquely identifies the Vendor of the Device. See Vendor ID.

1.4. Standards Terminology Mapping

Matter	HomeKit	Weave	Thread	Zigbee
Administrator	Admin	Fabric provisioner	Commissioner	Coordinator
Attribute	Characteristics	Property		Attribute

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Matter	HomeKit	Weave	Thread	Zigbee
Binding	Event subscription	Subscription	Link	Binding
Broadcast			Broadcast	Broadcast
Client		Service client	Client	Client
Cluster	Services	interface		Cluster
Cluster	Trait	Service		Cluster
Command	Command	Command	Command	Command
Commissioning	Pairing	Pairing	Commissioning	Association
Commissioner	Admin	Fabric provisioner	Commissioner	Coordinator
Device	Accessory	Device	Device	Device
End Device			End Device	End Device
Endpoint	Profile	Resource	Interface	Endpoint
Endpoint Address	Device ID	Resource ID	Endpoint Identi! fier	Endpoint address
Fabric	Network	Fabric	Partition	Network
Network Manager	Device / Controller	Nest Service	Leader	Network manager
Node	Accessory	Node	Node	Node
Router			Router	Router
Server		Service host	Server	Server
Service Discovery		Service directory		Service Discovery

1.5. Conformance Levels

The key words below are usually capitalized in the document to make the requirement clear.

Key Word	Description
MAY	A key word that indicates flexibility of choice with no implied preference.
NOT	A key word that used to describe that the requirement is the inverse of the behav! ior specified (i.e. SHALL NOT, MAY NOT, etc)
SHALL	A key word indicating a mandatory requirement. Designers are required to imple! ment all such mandatory requirements.
SHOULD	A key word indicating flexibility of choice with a strongly preferred alternative. Equivalent to the phrase is recommended.

1.6. References

The following standards and specifications contain provisions, which through reference in this doc! ument constitute provisions of this specification. All the standards and specifications listed are nor!

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

mative references. At the time of publication, the editions indicated were valid. All standards and specifications are subject to revision, and parties to agreements based on this specification are encouraged to investigate the possibility of applying the most recent editions of the standards and specifications indicated below.

1.6.1. CSA Reference Documents

Reference	Reference Location/URL	Description
[CSA-05-03874]	https://groups.csa-iot.org/wg/members-all/document/10905	CSA Manufacturer Code Database
[AppClusters]	https://github.com/CHIP-Specifications/connected!homeip-spec/raw/build-sam!ple/pdf/appclusters.pdf	Application Clusters - Under development
[Matter Brand Guide!lines]	https://groups.csa-iot.org/wg/members-all/document/22901	Matter Brand Guidelines

1.6.2. External Reference Documents

Reference	Reference Location/URL	Description
[AdProx]	https://tools.ietf.org/html/draft-sctl-advertising-proxy	Advertising Proxy for DNS-SD SRP
[ANSIÊC18]	https://ansi.org	ANSI C18 Standards on Portable Cells and Batteries
[Bluetooth¨]	https://www.bluetooth.org/docman/handlers/download!doc.ashx?doc_id=441541	Bluetooth¨ Core Specification 4.2
[FIPSÊ180-4]	https://csrc.nist.gov/publica!tions/detail/fips/180/4/final	NIST FIPS 180-4 Secure Hash Standard (SHS), August 2015
[FIPSÊ186-4]	https://csrc.nist.gov/publica!tions/detail/fips/186/4/final	NIST FIPS 186-4 Digital Signature Standard (DSS), July 2013
[FIPSÊ197]	https://doi.org/10.6028/NIST.FIPS.197	NIST FIPS 197 Advanced Encryption Standard (AES), November 2001
[FIPSÊ198-1]	https://csrc.nist.gov/publica!tions/detail/fips/198/1/final	NIST FIPS 198-1 The Keyed-Hash Message Authentica!tion Code (HMAC), July 2008
[IECÊ60086]	https:///www.iec.ch	IEC 60086 standard for Primary Batteries
[IEEEÊ754-2019]	https://ieeexplore.ieee.org/document/8766229	"IEEE Standard for Floating-Point Arithmetic," in IEEE Std 754-2019 (Revision of IEEE 754-2008) July 2019, doi: 10.1109/IEEESTD.2019.8766229.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Reference	Reference Location/URL	Description
[IEEEÊ802.11-2020]	https://standards.ieee.org/standard/802_11-2020.html	IEEE 802.11-2020 - IEEE Standard for Information Technology - Telecommunications and Information Exchange between Systems - Local and Metropolitan Area Networks - Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications
[ISO/IECÊ18004:2015]	https://www.iso.org/standard/62021.html	Information technology - Automatic identification and data capture techniques - QR Code bar code symbology specification
[ITUÊASN.1]	https://www.itu.int/en/ITU-T/asn1/Pages/asn1_project.aspx	ITU ASN.1 Project
[NFCForum-TS-NDEFÊ1.0]	https://nfc-forum.org/our-work/specification-releases/specifications/nfc-forum-technical-specifications	Data Exchange Format (NDEF) Technical Specification, NFC Forum
[NFCForum-TS-RTDÊ1.0]	https://nfc-forum.org/our-work/specification-releases/specifications/nfc-forum-technical-specifications/	Record Type Definition (RTD) Technical Specification, NFC Forum
[NFCForum-TS-RTDÊURIÊ1.0]	https://nfc-forum.org/our-work/specification-releases/specifications/nfc-forum-technical-specifications/	URI Record Type Definition Technical Specification, NFC Forum
[NISTÊ800-38A]	https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf	NIST SP 800-38A Recommendation for Block Cipher Modes of Operation: Methods and Techniques, December 2001
[NISTÊ800-38C]	https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38c.pdf	NIST SP 800-38C Recommendations for Block Cipher Mode of Operation: The CCM Mode for Authentication and Confidentiality, Morris Dworkin, May 2004 (errata update 2007)
[NISTÊ800-56C]	https://csrc.nist.gov/publications/detail/sp/800-56c/rev-2/final	NIST SP 800-56C Recommendation for Key-Derivation Methods in Key-Establishment Schemes, Revision 2, August 2020
[NISTÊ800-90A]	https://csrc.nist.gov/publications/detail/sp/800-90a/rev-1/final	NIST SP 800-90A Rev. 1 Recommendation for Random Number Generation Using Deterministic Random Bit Generators
[NISTÊ800-90B]	https://csrc.nist.gov/publications/detail/sp/800-90b/final	NIST SP 800-90B Recommendation for the Entropy Sources Used for Random Bit Generation
[NISTÊ800-132]	https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-132.pdf	NIST SP 800-132 Recommendation for Password-Based Key Derivation, Part 1: Storage Applications, December 2010

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Reference	Reference Location/URL	Description
[NISTÊ800-186]	https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-186-draft.pdf	NIST Draft SP 800-186 Recommendation for Discrete Logarithm-Based Cryptography, October 2019
[RFCÊ1350]	https://www.rfc-editor.org/rfc/rfc1350	The TFTP Protocol (Revision 2)
[RFCÊ2119]	https://www.rfc-editor.org/rfc/rfc2119	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997
[RFCÊ2782]	https://www.rfc-editor.org/rfc/rfc2782	A DNS RR for specifying the location of services (DNS SRV)
[RFCÊ2986]	https://www.rfc-editor.org/rfc/rfc2986	PKCS #10: Certification Request Syntax Specification Version 1.7
[RFCÊ3306]	https://www.rfc-editor.org/rfc/rfc3306	Unicast-Prefix-based IPv6 Multicast Addresses
[RFCÊ3587]	https://www.rfc-editor.org/rfc/rfc3587	IPv6 Global Unicast Address Format
[RFCÊ3986]	https://www.rfc-editor.org/rfc/rfc3986	Uniform Resource Identifier (URI)
[RFCÊ4007]	https://www.rfc-editor.org/rfc/rfc4007	IPv6 Scoped Address Architecture
[RFCÊ4191]	https://www.rfc-editor.org/rfc/rfc4191	Default Router Preferences and More-Specific Routes
[RFCÊ4193]	https://www.rfc-editor.org/rfc/rfc4193	Unique Local IPv6 Unicast Addresses (ULA)
[RFCÊ4291]	https://www.rfc-editor.org/rfc/rfc4291	IPv6 Addressing Architecture
[RFCÊ4506]	https://www.rfc-editor.org/rfc/rfc4506	XDR: External Data Representation Standard
[RFCÊ4648]	https://www.rfc-editor.org/rfc/rfc4648	The Base16, Base32, and Base64 Data Encodings
[RFCÊ4861]	https://www.rfc-editor.org/rfc/rfc4861	Neighbor Discovery for IP version 6 (IPv6)
[RFCÊ4862]	https://www.rfc-editor.org/rfc/rfc4862	IPv6 Stateless Address Autoconfiguration
[RFCÊ5280]	https://www.rfc-editor.org/rfc/rfc5280	Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile
[RFCÊ5505]	https://www.rfc-editor.org/rfc/rfc5505	Principles of Internet Host Configuration

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Reference	Reference Location/URL	Description
[RFCÊ5652]	https://www.rfc-editor.org/rfc/rfc5652	Cryptographic Message Syntax (CMS)
[RFCÊ6335]	https://www.rfc-editor.org/rfc/rfc6335	Service Name and Port Number Procedures
[RFCÊ6760]	https://www.rfc-editor.org/rfc/rfc6760	Replacement of AppleTalk NBP
[RFCÊ6762]	https://www.rfc-editor.org/rfc/rfc6762	Multicast DNS
[RFCÊ6763]	https://www.rfc-editor.org/rfc/rfc6763	DNS-Based Service Discovery
[RFCÊ6920]	https://www.rfc-editor.org/rfc/rfc6920	Naming Things with Hashes
[RFCÊ7230]	https://www.rfc-editor.org/rfc/rfc7230	Hypertext Transfer Protocol (HTTP/1.1): Message Syn! tax and Routing
[RFCÊ7346]	https://www.rfc-editor.org/rfc/rfc7346	IPv6 Multicast Address Scopes
[RFCÊ7468]	https://www.rfc-editor.org/rfc/rfc7468	Textual Encodings of PKIX, PKCS, and CMS Structures
[RFCÊ7558]	https://www.rfc-editor.org/rfc/rfc7558	Scalable DNS-SD Requirements
[RFCÊ8305]	https://www.rfc-editor.org/rfc/rfc8305	Happy Eyeballs Version 2: Better Connectivity Using Concurrency
[RFCÊ8490]	https://www.rfc-editor.org/rfc/rfc8490	DNS Stateful Operations
[RFCÊ8765]	https://www.rfc-editor.org/rfc/rfc8765	DNS Push Notifications
[RFCÊ8766]	https://www.rfc-editor.org/rfc/rfc8766	Discovery Proxy
[draft-lemon-stub-net! works]	https://datatracker.ietf.org/doc/html/draft-lemon-stub-networks-02	Connecting Stub Networks to Existing Infrastructure
[SECÊ1]	https://www.secg.org/sec1-v2.pdf	SEC 1: Elliptic Curve Cryptography, Version 2.0, Certi! com Research, May 2009
[SECÊ2]	https://secg.org/sec2-v2.pdf	SEC 2: Recommended Elliptic Curve Domain Parame! ters, Version 2.0, Certicom Research, January 2010

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Reference	Reference Location/URL	Description
[SIGMA]	https://doi.org/10.1007/978-3-540-45146-4_24	Krawczyk H. (2003) SIGMA: The ÔSIGn-and-MAcÕ Approach to Authenticated Diffie-Hellman and Its Use in the IKE Protocols. In: Boneh D. (eds) Advances in Cryptology - CRYPTO 2003. CRYPTO 2003. Lecture Notes in Computer Science, vol 2729. Springer, Berlin, Heidelberg.
[SPAKE2+]	https://tools.ietf.org/pdf/draft-bar-cfrg-spake2plus-02.pdf	SPAKE2+, an Augmented PAKE (Draft 02, 10 Decem! ber 2020)
[SRP]	https://tools.ietf.org/html/draft-ietf-dnssd-srp	Service Registration Protocol
[Thread]	https://www.thread! group.org	Thread 1.3.0 Specification
[Verhoeff]	https://ir.cwi.nl/pub/13045	Verhoeff, J. (1969). Error detecting decimal codes. MC Tracts. Centrum Voor Wiskunde en Informatica.
[X.501]	https://www.itu.int/rec/T-REC-X.501/en	ITU X.501 : Information technology - Open Systems Interconnection - The Directory: Models
[X.509]	https://www.itu.int/rec/T-REC-X.509/en	ITU X.509 : Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks
[X.520]	https://www.itu.int/rec/T-REC-X.520/en	ITU X.520 : Information technology - Open Systems Interconnection - The Directory: Selected attribute types
[X.680]	https://www.itu.int/rec/T-REC-X.680/en	ITU X.680 : Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation
[X.690]	https://www.itu.int/rec/T-REC-X.690/en	ITU X.690 : Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)

1.7. Informative References

1.7.1. CSA Reference Documents

Reference	Reference Location/URL	Description
[DotdotArch]	https://groups.csa-iot.org/wg/matter-tsg/document/18649	Dotdot Architecture Model, document 13-0589, revi! sion 14, February 2019
[ZCL]	https://groups.csa-iot.org/wg/members-all/document/23019	Zigbee Cluster Library Specification, document 07-5123, revision 8, December 2019

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Reference	Reference Location/URL	Description
[CSA-PNP]	https://groups.csa-iot.org/wg/members/document/21624	Organizational Processes and Procedures, 13-0625, revision 8, November 2021

1.8. Conventions

The following conventions are used in this document.

1.8.1. Enumerations and Reserved Values

An undefined value or range of an enumeration, field, or identifier SHALL be considered reserved for future revisions of this standard and SHALL not be available for implementation.

A value or range of an enumeration, field, or identifier that is available for non-standard imple! mentation SHALL be described as Òmanufacturer specificÓ, ÒmsÓ, or ÒMSÓ.

A value or range of an enumeration, field, or identifier that is available for other parts of this stan! dard SHALL be described as such.

A value or range of an enumeration, field, or identifier that is deprecated, and not available for implementation, SHALL be described as ÒDeprecatedÓ or ÒDÓ.

1.8.2. Reserved Bit Fields

Each full or partial data field (e.g., message data field), of any bit length, that is undefined, SHALL be considered reserved for future revisions of this standard and SHALL not be available for imple! mentation.

An implementation of a revision where a bit is reserved SHALL indicate that bit as zero when con! veying that bit in a message, and ignore that bit when conveyed from another implementation.

1.8.3. Number Format

In this specification, hexadecimal numbers are prefixed with the designation Ò0xÓ and binary num! bers are prefixed with the designation Ò0bÓ. All other numbers are assumed to be decimal unless indicated otherwise within the associated text.

Binary numbers are specified as successive groups of 4 bits, separated by a space (Ò Ò) character from the most significant bit (next to the 0b prefix and leftmost on the page) to the least significant bit (rightmost on the page), e.g. the binary number 0b0000 1111 represents the decimal number 15. Where individual bits are indicated (e.g. "bit 3") the bit numbers are relative to the least significant bit which is bit 0.

When a bit is specified as having a value of either 0 or 1 it is specified with an ÒxÓ, e.g. Ò0b0000 0xxxÓ indicates that the lower 3 bits can take any value but the upper 5 bits must each be set to 0.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

1.8.4. Provisional

Per [CSA-PNP], when a specification is completed there may be sections of specification text (or smaller pieces of a section) that are not certifiable at this stage. These sections (or smaller pieces of a section) are marked as provisional prior to publishing the specification. This specification uses well-defined notation to mark Provisional Conformance or notes a section of text with the term "provisional".

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Chapter 2. Architecture

2.1. Overview

Matter aims to build a universal IPv6-based communication protocol for smart home devices. The protocol defines the application layer that will be deployed on devices as well as the different link layers to help maintain interoperability. The following diagram illustrates the normal operational mode of the stack:

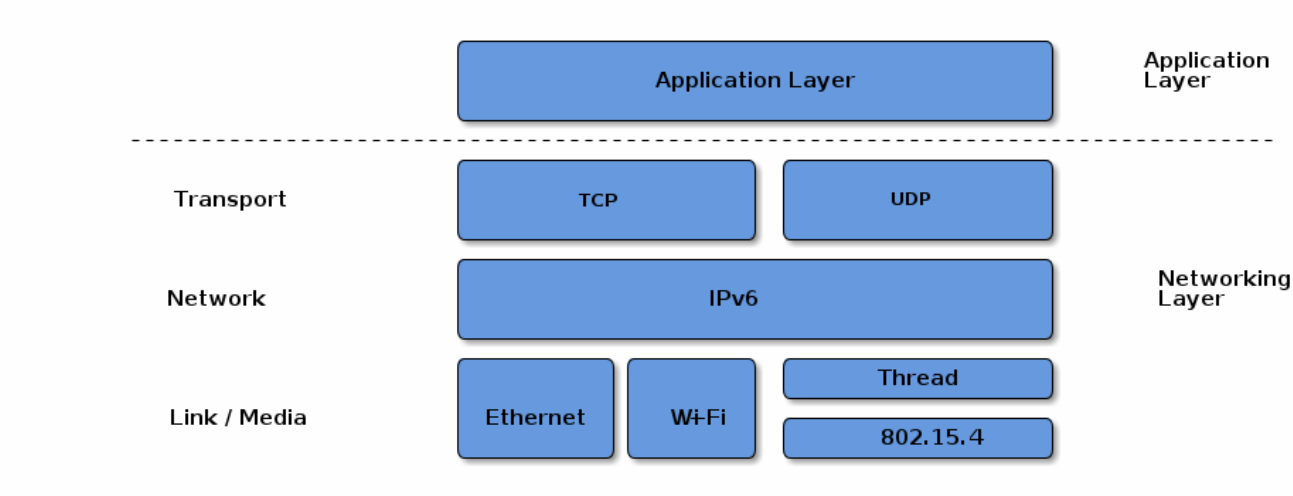


Figure 1. Application and Network Stack

2.2. Layered Architecture

The architecture is divided into layers to help separate the different responsibilities and introduce a good level of encapsulation amongst the various pieces of the protocol stack. The vast majority of interactions flow through the stack captured in the following Figure .

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

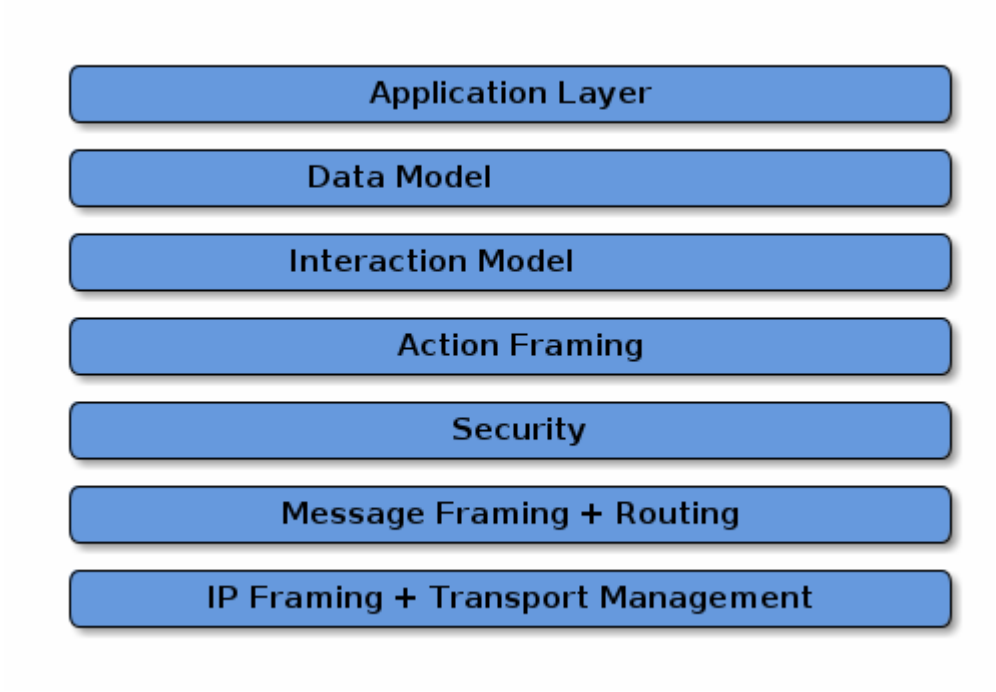


Figure 2. Layered Architecture

The Application layer corresponds to the high order business logic of a device. For example, an application that is focused on lighting might contain logic to handle turning on/off a light bulb, as well as its color characteristics.

The Data Model layer corresponds to the data and verb elements that help support the functionality of the application. The Application operates on these data structures when there is intent to interact with the device.

The Interaction Model layer defines a set of interactions that can be performed between a client and server device. For example, reading or writing attributes on a server device would correspond to application behavior on the device. These interactions operate on the elements defined at the data model layer.

Once an action is constructed using the Interaction Model , it is serialized into a prescribed packed binary format to encode for network transmission. This process is handled in the Action Framing layer.

An encoded action frame is then processed by the Security Layer : the message is encrypted and appended with a message authentication code. These actions ensure the data remain confidential and authentic between sender and receiver of the message.

With an interaction now serialized, encrypted, and signed, the Message Layer constructs the pay! load format with required and optional header fields, which specify properties of the message as well logical routing information.

After the final payload has been constructed by the Message Layer, it is sent to the underlying trans! port protocol (TCP or MatterÕs Message Reliability Protocol) for IP management of the data.

Once the data is received on the peer device, it travels up the protocol stack, where the various lay! ers reverse the operations on the data performed by the sender, to finally deliver the message to the Application for consumption.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

In addition to the data flows captured above, this specification defines secure session establishment protocols based on operational certificates (see Section 4.13.2, ÒCertificate Authenticated Session Establishment (CASE)Ó), or passcodes (see Section 4.13.1, ÒPasscode-Authenticated Session Establishment (PASE)Ó), group communication (see Section 4.14, ÒGroup CommunicationÓ), a bulk data transfer protocol (BDX) suitable for sending bulk data between Nodes, and provisions for defining manufacturer-specific protocols.

2.3. Network Topology

In principle, any IPv6-bearing network is suitable for Matter deployment, subject to supporting a few core IPv6 standards. In this version of the specification, we focus on three link layer technologies: Ethernet, Wi-Fi and Thread. We restrict the specification to the above so that the specification can suitably cover provisioning of these link layers, and so that the amount of testing in certification is suitably bounded.

Matter treats networks as shared resources: it makes no stipulation of exclusive network ownership or access. As a result, it is possible to overlay multiple Matter networks over the same set of constituent IP networks.

This protocol may operate in the absence of globally routable IPv6 infrastructure. This requirement enables operation in a network disconnected or firewalled from the global Internet. It also enables deployment in situations where the Internet Service Provider either does not support IPv6 on consumer premises or where the support proves otherwise limiting, for example, if the delegated prefix cannot accommodate all the networks and devices on premises.

This protocol supports local communications spanning one or more IPv6 subnets. Canonical networks supporting a fabric may include a Wi-Fi/Ethernet subnet, or one or more low power and lossy networks (LLN) subnets. In this version of the specification, Thread is the supported LLN standard.

2.3.1. Single network

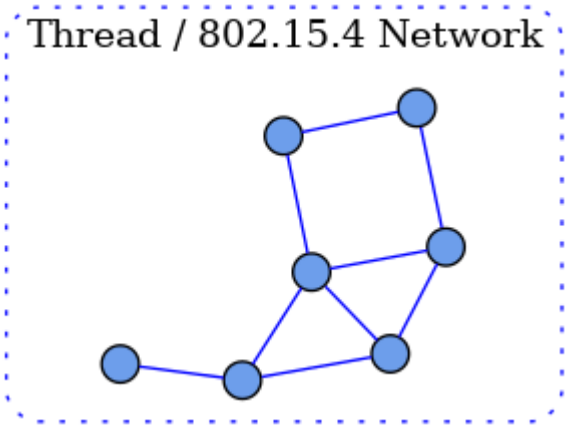


Figure 3. Single Thread network

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

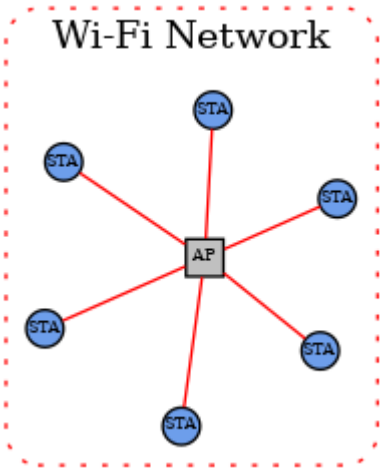
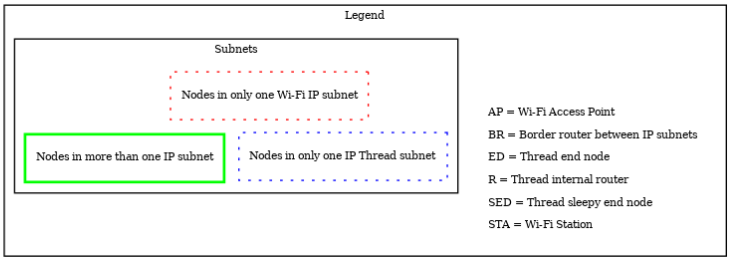


Figure 4. Single Wi-Fi network

In the single network topology, all Matter devices are connected to a single logical network. This could be a Thread/802.15.4 network, a Wi-Fi network or an Ethernet network. In the case of Wi-Fi/Ethernet, the network could in fact span multiple Wi-Fi and/or Ethernet segments provided that all the segments are bridged at the link layer. A Node is a single instance of a Matter device within a fabric, operationally available on an IP network.

Each Node in the single-network topology communicates with every other Node in the Fabric via a single network interface.

2.3.2. Star network topology



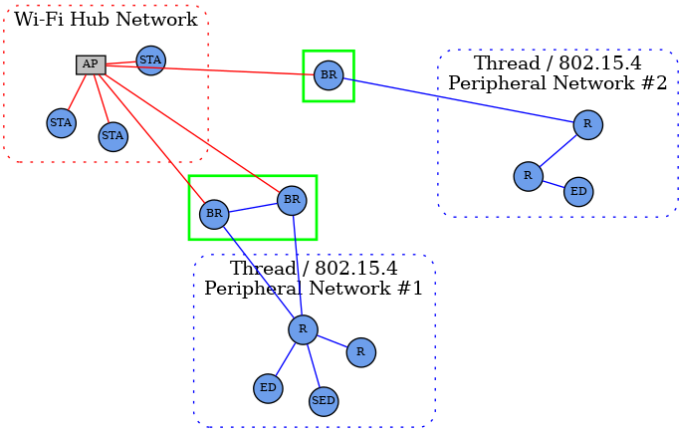


Figure 5. Star network topology

The star network topology consists of multiple peripheral networks joined together by a central hub network. The hub network will typically be the customerÕs home network (Wi-Fi/Ethernet net! work), while the peripheral networks can be of any supported network type. A peripheral network MUST always be joined directly to the hub network via one or more Border Routers.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Architecturally, any number of peripheral networks may be present in a single Fabric, including multiple networks of the same type. Nodes MAY have interfaces onto any network (hub or periph! eral), and MAY communicate directly to other Nodes on the same network. However, any communi! cation that must cross a network boundary to reach its destination MUST flow through a Border Router.

This protocol places a set of requirements on the Border Router. These requirements pertain to address assignment, route assignment and advertisement, multicast support, and discovery proxy! ing.

Note that in this version of the specification, Thread is the primary supported LLN. In many cases, customer installations will attempt to maintain a simple network topology, with a single Wi-Fi/Eth! ernet subnet, and a single Thread network. However, more than one Thread network is possible and supported.

To support home automation interoperability, this protocol supports the concept of bridging which makes available, through a data model node, devices implementing other home automation tech! nologies, transports and link layers.

2.4. Scoped names

The Matter protocol explicitly supports multiple administrators, unrelated by any common roots of trust (multi-admin). This functionality is addressed via multiple fabrics and is enabled by the core aspects of name scoping (see below), and key considerations enabling multiple fabrics in onboard! ing , secure communication , and aspects of the data model (such as fabric-scoped data).

A Fabric is a collection of Matter devices sharing a trusted root. The root of trust in Matter is the Root CA that issues the NOCs which underpin node identities. Within the fabric, each node is uniquely identified by a stable Node ID. The scoped selection and allocation of these constructs within Matter ensures the uniqueness of identifiers and gives clear guidance on ownership and management of namespaces.

The operational root of trust$Ñ$the root certificate authority (CA) as identified by its public key (Root PK)$Ñ$is responsible for the allocation of correctly scoped fabric identifiers. The security of all public key infrastructures (PKI) depends on the private key of the CA being protected and neither guessable nor obtainable; that property, in turn, implies that the public key is globally unique. Within any root CA, the fabrics$Ñ$identified by a 64-bit number$Ñ$are unique. The uniqueness mechanism emerges from the collaboration of the commissioner and the root CA associated with that particular commissioner. Matter wraps the collaboration between the commissioner and its associated root CA and other possible data stores into a concept called the "administrative domain manager" (ADM). The algorithmic details and policies within the administrative domain manager are out of the scope of the specification as long as the allocation of all identifiers obeys the unique! ness and scoping criteria. Fabrics are uniquely identified by the tuple of their root CAÕs public key and a Fabric ID. Similarly, within each fabric, the administrative domain manager is responsible for assigning a unique and stable Operational Node ID to every node.

The scoping strategy as outlined here ensures that each scoped identifier can be allocated solely by the entities responsible for the scoping, without consideration for collisions or forgeries. For exam! ple, two different CAs may allocate the same fabric identifiers and this would not create any prob!

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

lems for the devices within the network. Scoped delegation of responsibility also provides for clear guidelines for the removal of specific identifiers.

A Matter device may be a member of multiple fabrics and thus have multiple associated node IDs. The scoping strategy also naturally lends itself towards unambiguous resolution of names and cre! dentials and places a clearly defined responsibility for managing the namespaces on each fabricÕs associated administrative domain manager service.

Prior to the first commissioning, such as in factory-reset state, a typical device contains no pre-allo! cated operational roots of trust, and no operational identities in the form of fabric IDs and node IDs. Yet, various interactions expect the fabric ID, or a node ID. These identifiers emerge in a num! ber of internal constructs$Ñ$from address discovery, through identifying secure sessions, to evaluat! ing access control privileges. In order to regularize all interactions with the device and solve the bootstrapping problem, a special primordial fabric ID is reserved, and associates a set of initial access control privileges with any communication that would be associated with the initial commis! sioning sessions.

2.5. Identifiers

2.5.1. Fabric References and Fabric Identifier

As described above, a Fabric ID is a 64-bit number that uniquely identifies the Fabric within the scope of a particular root CA. Conceptually, the fully qualified fabric reference consists of the tuple containing the public key of the root certificate authority, and the Fabric ID. Because the fully quali! fied fabric reference is cumbersome to use, a number of mechanisms for compression of the refer! ence are defined. The Fabric reference, in compressed form, is used during operational discovery to provide operational naming separation, a form of namespacing, between unrelated collections of devices.

Fabric ID 0 is reserved across all fabric root public key scopes. This fabric ID SHALL NOT be used as the identifier of a fabric.

A fabric is defined in the Data Model as a set of nodes that interact by accessing Data Model ele! ments as defined in the Interaction Model (see Section 7.5, ÒFabricÓ).

The layers below the data model, that convey data model interactions as messages, SHALL always indicate either the fabric associated with the message, or that there is no fabric associated with the message.

For example: A Data Model message that is conveyed over a message channel that uses the reserved fabric ID '0' does not have a fabric associated with it.

2.5.2. Vendor Identifier (Vendor ID, VID)

A Vendor Identifier (Vendor ID or VID) is a 16-bit number that uniquely identifies a particular prod! uct manufacturer, vendor, or group thereof. Each Vendor ID is statically allocated by the Connectiv! ity Standards Alliance (see [CSA Manufacturer Code Database]).

The following Vendor IDs are reserved:

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Table 1. Vendor ID Allocations

Range	Type
0x0000	Matter Standard
0x0001 - 0xFFF0	reserved for individual Manufacturer Codes as per CSA Manufacturer Code Database
0xFFF1	Test Vendor #1
0xFFF2	Test Vendor #2
0xFFF3	Test Vendor #3
0xFFF4	Test Vendor #4

All other allocations of Vendor ID are specified in CSA Manufacturer Code Database .

NOTE

The Test Vendor IDs are reserved for test and development by device manufacturers or hobbyists. Commissioners SHOULD NOT commission devices using one of these VIDs onto an operational Fabric under normal operation unless the user is made fully aware of the security risks of providing an uncertified device with operational and networking credentials.

2.5.3. Product Identifier (Product ID, PID)

A Product Identifier (Product ID or PID) is a 16-bit number that uniquely identifies a product of a vendor. The Product ID is assigned by the vendor and SHALL be unique for each product within a Vendor ID . Once a Product ID has been used, it SHALL NOT be reused by a different product type under the same Vendor ID. These Product IDs SHOULD NOT be specific to a unique physical device; rather they describe the product type, which might have many manufactured instances (e.g. multi! ple colors of the same product type).

A value of 0x0000 SHALL NOT be assigned to a product since Product ID = 0x0000 is used for these specific cases:

- ¥ To announce an anonymized Product ID as part of device discovery (see Section 5.4.2, ÒAnnouncement by DeviceÓ).
- ¥ To indicate an OTA software update file applies to multiple Product IDs equally.
- ¥ To avoid confusion when presenting the Onboarding Payload for ECM with multiple nodes .

2.5.4. Group Identifier (GID)

A Group Identifier (Group ID or GID) is a 16-bit number that identifies a set of Nodes across a Fabric at the message layer (see Section 4.15, ÒGroup Key ManagementÓ). A Group ID can further be bound to one or more Endpoints within each Node in the group at the interaction layer.

The Group ID space is allocated as described in Table 2, ÒGroup ID AllocationsÓ :

Table 2. Group ID Allocations

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Range	Type
0xFF00 - 0xFFFF	Universal Group ID range reserved for static multi! cast and anycast identifiers
0x0001 - 0xFEFF	Application Group ID, assigned by fabric Administra! tor
0x0000	Null or unspecified Group ID

2.5.4.1. Universal Group ID

A Universal Group ID (UGID) is one that resides in the 16-bit subrange of Group ID that is reserved for groups that are common across this standard. These special multicast, groupcast, or anycast des! tinations are constant and known to all Nodes on any Fabric. The Universal Group ID space is allo! cated as described in Table 3, ÒUniversal Group ID AllocationsÓ :

Table 3. Universal Group ID Allocations

Range	Type
0xFFFF	All Nodes
0xFFFE	All non-sleepy Nodes
0xFFFD	All Proxies
0xFF00-0xFFFC	Reserved for future use

The Commissioner SHALL configure one or more shared keys for these groups on all Nodes within the Fabric. Because the keys and IPv6 multicast prefixes are different across Fabrics, Universal Groups only enable communication within a specific Fabric.

All Nodes Group

This group is used to message all Nodes in a Fabric.

All non-sleepy Nodes Group

This group is used to message all power-capable Nodes in a Fabric. Sleepy Nodes SHALL NOT sub! scribe to this group.

All Proxies Group

This group is used to discover Proxy Nodes during Section 9.15.4, ÒProxy SubscriptionsÓ.

2.5.5. Node Identifier

A Node Identifier (Node ID) is a 64-bit number that uniquely identifies an individual Node or a group of Nodes on a Fabric. The Node Identifier space is allocated as described in Table 4, ÒNode Identifier AllocationsÓ :

Table 4. Node Identifier Allocations

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Range	Type
0xFFFF_FFFF_FFFF_xxxx	Group Node ID
0xFFFF_FFFF_0000_0000 to 0xFFFF_FFFF_FFFE_FFFF	Reserved for future use
0xFFFF_FFFE_xxxx_xxxx	Temporary Local Node ID
0xFFFF_FFFD_xxxx_xxxx	CASE Authenticated Tag
0xFFFF_FFFC_xxxx_xxxx to 0xFFFF_FFFC_FFFF_FFFF	Reserved for future use
0xFFFF_FFFB_xxxx_xxxx	PAKE key identifiers
0xFFFF_FFF0_0000_0000 to 0xFFFF_FFFA_FFFF_FFFF	Reserved for future use
0x0000_0000_0000_0001 to 0xFFFF_FFFE_FFFF_FFFF	Operational Node ID
0x0000_0000_0000_0000	Unspecified Node ID

Node IDs are used for core messaging, within the internal APIs, within the data model, and to resolve the operational IPv6 addresses of Nodes (see Section 4.3.2, ÒOperational DiscoveryÓ).

The span of Node IDs from 0xFFFF_FFF0_0000_0000 to 0xFFFF_FFFF_FFFF_FFFF, as well as the value 0x0000_0000_0000_0000 are both reserved for special uses.

2.5.5.1. Operational Node ID

An Operational Node ID is a 64-bit number that uniquely identifies an individual Node on a Fabric. All messages must have an Operational Node ID as the source address. All unicast messages must have an Operational Node ID as the destination address.

While source or destination address MAY be elided from a message, it MUST remain unambiguously derivable from the Session ID.

2.5.5.2. Group Node ID

A Group Node ID is a 64-bit Node ID that contains a particular Group ID in the lower half of the Node ID.

2.5.5.3. Temporary Local Node ID

A Temporary Local Node ID is a 64-bit Node ID that contains an implementation-dependent value in its lower 32 bits. This allows implementations to keep track of connections or transport-layer links and similar housekeeping internal usage purposes in contexts where an Operational Node ID is unavailable.

2.5.5.4. PAKE key identifiers

This subrange of Node ID is used to assign an access control subject to a particular PAKE key as specified in Section 6.6.2.1.1, ÒPASE and Group SubjectsÓ. An example usage would be to create an

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

ACL entry to provide administrative access to any commissioner communicating via a PASE session established with a particular pincode.

2.5.5.5. CASE Authenticated Tag

This subrange of Node ID is used to assign an access control subject to a group of peer nodes that share a single CASE session as specified in Section 6.6.2.1.2, ÒSubjects identified by CASE Authenti! cated TagÓ.

2.5.5.6. Unspecified Node ID

The Unspecified Node ID (0x0000_0000_0000_0000) is a reserved value that never appears in mes! sages or protocol usage. It exists to mark or detect the presence of uninitialized, missing, or invalid Node IDs.

2.5.6. IPv6 Addressing

This protocol uses IPv6 addressing for its operational communication. Node IDs and Fabric IDs are resolved to various types of IPv6 addresses [RFCÊ4291].

2.5.6.1. IPv6 Unicast Address

An IPv6 Unicast Address is one that uniquely identifies and addresses a single Node on an IPv6 net! work. A primary design goal for this standard is to allow Nodes to leverage native IPv6 technolo! gies. As such, an operational IPv6 Unicast address that provides connectivity and routability between Nodes SHALL be used. This includes a global unicast address (GUA), a link-local address (LLA), or a unique local address (ULA).

2.5.6.2. IPv6 Multicast Address

An IPv6 Multicast Address is formed using Unicast-Prefix-based IPv6 Multicast Addresses [RFCÊ3306].

- ¥ The first 12 bits are defined by [RFCÊ3306] and are 0xFF3.
- ¥ The next 4 bits are "scop" (scope) and set based on [RFCÊ7346] Section 2 to:
  - %Site-Local (0x5) - spans all networks in the Fabric, including Thread, Ethernet, and Wi-Fi.
- ¥ The next 8 bits are reserved (0x00).
- ¥ The next 8 bits are "plen", and set to 0x40 to indicate a 64-bit long network prefix field.

The network prefix portion of the Multicast Address is the 64-bit bitstring formed by concatenating:

- ¥ 0xFD to designate a locally assigned ULA prefix per [RFCÊ4193] Section 3.1
- ¥ The upper 56-bits of the Fabric ID for the network in big-endian order

The 32-bit group identifier portion of the Multicast Address is the 32-bits formed by:

- ¥ The lower 8-bits of the Fabric ID
- ¥ 0x00

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

¥ The next 16-bits are the Group Identifier for the group, as specified in Group Identifier in big-endian order

An example of the site local scoped multicast address for a given <Fabric ID> and <Group ID>:

Ê FF35:0040:FD<Fabric ID>00:<Group ID>

NOTE though Site-Local scope is always used, the effective scope MAY be limited by setting the IPv6 hop count.

The Multicast Address formation ensures a low probability of a node receiving a multicast message it is not interested in. If a collision does occur on the multicast address (which requires two identi! cal 64-bit Fabric IDs and two identical 16-bit Group IDs), processing of the message disambiguates which fabric and group is relevant by checking which operational group key leads to the messageÕs 64-bit MIC.

2.5.6.3. IPv6 Multicast Port Number

The IANA assigned port number is 5540

2.5.6.4. IPv4 Coexistence

Matter devices SHALL be tolerant of IPv4 addresses and MAY ignore those addresses for the pur! poses of Matter operations.

2.6. Device identity

Each Matter device holds a number of certificate chains.

A Device Attestation Certificate (DAC) proves the authenticity of the manufacturer and a certifica! tion status of the deviceÕs hardware and software. The Device Attestation Certificate is used during the commissioning process by the Commissioner to ensure that only trustworthy devices are admit! ted into a Fabric. The details of the device attestation process are captured in Section 6.2, ÒDevice AttestationÓ.

Each Matter device is issued an Operational Node ID and a Node Operational Certificate (NOC) for that Operational Node ID. The NOC enables a Node to identify itself within a Fabric by cryptographi! cally binding a unique Node Operational Key Pair to the operational identity of its subject, attestable through the signature of a trusted Certificate Authority (CA). Operational Node IDs are removed on factory reset or removal of Fabrics. A NOC is issued during the commissioning process of a device into a Fabric. These steps help to protect the privacy of the end-user and to adapt to dif! ferent trust models.

The format of the Node Operational credentials and protocols for generating those credentials are detailed in Section 6.4, ÒNode Operational Credentials SpecificationÓ and Section 6.5, ÒOperational Certificate EncodingÓ sections.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

2.7. Security

Matter deploys modern security practices to protect the Fabric. Matter designates a core set of secu! rity primitives detailed in Chapter 3, Cryptographic Primitives to provide comprehensive protection. Elliptic curve cryptography, based on the NIST P-256 curve (secp256r1) serves as the foundation for public key cryptography and digital signatures. Commonly available AES modes of operation have been selected to provide shared key cryptographic operations. In scenarios involving an out-of-band passcode-based authentication, Matter uses SPAKE2+, an efficient augmented PAKE algorithm.

The core cryptographic primitives form the basis of a number of complementary secure protocols used within Matter. All unicast Node-to-Node messages are secured, authenticated, and provide replay protection. Building on top of IPv6 multicast, Matter also provides group messaging facilities, useful for efficiently addressing on an LLN. The group messaging features prioritize low latency of packet processing.

2.8. Device Commissioning

Device commissioning (see Chapter 5, Commissioning) is the process of joining a device to a Fabric. The device being commissioned is referred to as the Commissionee and the device administering commissioning is referred to as the Commissioner. Device commissioning consists of the following steps:

1. Device discovery (see Section 5.4, ÒDevice DiscoveryÓ and see Section 5.1, ÒOnboarding Pay! loadÓ): The Commissioner discovers commissionable devices on network interfaces such as Bluetooth Low Energy, Wi-Fi, or other connected IP network. The Commissioner obtains the out-of-band secret (Passcode) from the commissionable deviceÕs QR Code, Manual Pairing Code , NFC Tag or other means. This secret is used by Passcode-Authenticated Session Establishment (PASE) to establish a secure commissioning session. The order of discovering commissionable devices and obtaining the out-of-band secret from commissionable device is not critical.
2. Security setup with PASE (see Section 4.13.1, ÒPasscode-Authenticated Session Establishment (PASE)Ó): Establish encryption keys between the Commissioner and Commissionee using PASE. All messages exchanged between the Commissioner and Commissionee are encrypted using these PASE-derived keys. The process also establishes an attestation challenge used during the device attestation procedure .
3. Device attestation verification (see Section 6.2, ÒDevice AttestationÓ): Commissioner establishes the authenticity of the Commissionee as a certified device, notifying the user if the device is not certified.
4. Information configuration (see Section 6.4, ÒNode Operational Credentials SpecificationÓ , Sec! tion 11.9, ÒGeneral Commissioning ClusterÓ and Section 11.17, ÒNode Operational Credentials ClusterÓ): The Commissioner provides Commissionee with information such as regulatory domain, UTC time, Operational Certificate and network interfaces configuration.
5. Join network (see Section 11.8, ÒNetwork Commissioning ClusterÓ and Section 4.3.2, ÒOpera! tional DiscoveryÓ): The Commissioner triggers the Commissionee to connect to the operational network unless the Commissionee is already on the operational network. The NodeÕs/Commis! sioneeÕs IPv6 address is then either used (if already known) or discovered (if not known) by the Commissioner or Administrator.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

6. Security setup with CASE (see Section 4.13.2, ÒCertificate Authenticated Session Establishment (CASE)Ó): Derive encryption keys used to establish secure communication between the Commissioner or Administrator and Node with CASE. All unicast messages between a Commissioner or Administrator and a Node are encrypted using these CASE-derived keys.
7. Commissioning complete message exchange (see Section 11.9, ÒGeneral Commissioning ClusterÓ): A message exchange encrypted using CASE-derived encryption keys on the operational network that indicates successful completion of commissioning process.

A commissioner can reconfigure the Commissionee multiple times over the operational network after the commissioning is complete or over the commissioning channel after PASE-derived encryption keys are established during commissioning. The commissioning flows are described in Section 5.5, ÒCommissioning FlowsÓ.

2.9. Sleepy End Device (SED)

One goal of this standard is to provide support for low energy Nodes running on limited power sources such as batteries or limited energy scavenging. The Sleepy End Device (SED) operating mode is defined to help extend and optimize battery lifetimes for such Nodes. The SED operating mode mirrors and aligns with Thread SED behavior, but may be leveraged over other supported IP interfaces, including Wi-Fi. The steady state behavior of a SED Node is to disable its IP interface (and underlying radio or link technology). A SED then periodically wakes to communicate with some infrastructure device in order to participate on the network. In the case of a Thread network (see [Thread specification]]), the infrastructure device is a parent Thread Router. For Wi-Fi, the access point provides the required infrastructure support. Two intervals are defined:

- Idle mode , or slow-polling , sets the maximum time a SED will sleep before polling. This parameter affects both the minimum power consumption and maximum latency. The SLEEPY_IDLE_INTERVAL parameter communicates the maximum sleep interval of a node in idle mode.
- Active mode sets the SED into a fast-polling interval for maximum responsiveness when the Node is engaged in ongoing communication, such as an active Exchange. The SLEEPY_ACTIVE_INTERVAL parameter communicates the maximum sleep interval of a node in active mode.

A SED SHALL indicate it is a sleepy device to peer nodes by setting its SLEEPY_IDLE_INTERVAL to a value greater than the default and advertising that value per [Discovery_SII] .

NOTE	Because parent infrastructure devices have limited buffering space to cache messages on behalf of sleepy devices, SED communication patterns SHOULD be designed such that the SED is predominantly the initiator.

A Node determines whether it is in Active or Idle mode based on whether it has any outstanding Exchanges in the Message Layer. While there are Exchanges active, a node will remain in Active mode and poll at the fast-polling interval if it is a SED. Once all Exchanges are closed, a node SHOULD transition into Idle mode and poll at the slow-polling interval if it is a SED and the node has no other outstanding reasons for staying awake.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

2.10. Data Model Root

¥ Endpoint 0 (zero) SHALL be the root node endpoint .

¥ Endpoint 0 (zero) SHALL support the Root Node device type.

2.11. Stack Limits

2.11.1. System Model Limits

2.11.1.1. Access Control Limits

¥ A node SHALL guarantee that there are at least three Access Control Entries available for every fabric supported by the node.

For example: A node that supports 6 fabrics must support at least 18 ACL entries, and if it sup! ports N entries must enforce that any K fabrics together do not use more than N - 3*(6-K) entries.

¥ Device types MAY impose additional constraints on the number of ACL entries that need to be supported.

2.11.1.2. Group Limits

¥ A node SHALL support at least one group key per fabric for managing the IPK.

¥ If the node implements one or more device types with support for the Groups cluster, the node SHALL additionally support the maximum number of the required groups as specified by all of these implemented device types.

¥ A node SHALL support one IPv6 multicast group membership for every operational group it supports.

¥ Support for GroupKeyMulticastPolicy field in GroupKeySetStruct is provisional.

2.11.2. Interaction Model Limits

2.11.2.1. Read Interaction Limits

¥ A server SHALL ensure that every fabric the node is commissioned into can handle a single Read Interaction from a client on that fabric containing up to 9 paths.

¥ A server MAY permit Read Interactions even when there is no accessing fabric, subject to avail! able resources (e.g over PASE).

2.11.2.2. Subscribe Interaction Limits

¥ A publisher SHALL ensure that every fabric the node is commissioned into can support at least three Subscribe Interactions to the publisher and that each subscription SHALL support at least 3 attribute/event paths.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

- A server MAY permit Subscribe Interactions even when there is no accessing fabric, subject to available resources (e.g over PASE).
- Device type specifications MAY require a larger number of supported subscriptions or paths.
- SUBSCRIPTION_MAX_INTERVAL_PUBLISHER_LIMIT defines the upper limit for the publisher-selected maximum interval for any subscription. This SHALL be set to 60 minutes.
- The minimal supported capabilities, subject to the minimal constraints above, are reported in the CapabilityMinima attribute of the Basic Information cluster .

2.11.2.3. Invoke Interaction Limits

- An Invoke Request action SHALL be limited to a single concrete command path .

2.12. List of Provisional Items

The following is a list of provisional items.

2.12.1. Invoke Multiple Paths

- Support for an Invoke Interaction with multiple paths or wildcard paths is provisional.

2.12.2. EventList Global Attribute

The EventList global attribute is provisional.

2.12.3. Proxy Service

The Proxy Architecture , the Proxy Config and Proxy Discovery clusters are provisional.

2.12.4. Time Synchronization

The Time Synchronization feature is provisional.

2.12.5. Parameters and Constants

Table 5, ÒGlossary of parametersÓ is a glossary of parameters used in this chapter with a brief description for each parameter. A Node SHALL use the provided default value for each parameter unless the message recipient Node advertises an alternate value for the parameter via Operational Discovery.

Table 5. Glossary of parameters

Parameter Name	Description	Default Value
SLEEPY_IDLE_INTERVAL	Maximum sleep interval of node when in idle mode.	300 millisec! onds
SLEEPY_ACTIVE_INTERVAL	Maximum sleep interval of node when in active mode.	300 millisec! onds

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Parameter Name	Description	Default Value
SLEEPY_ACTIVE_THRESHOLD	Minimum amount the node SHOULD stay awake after network activity.	4000 millisec! onds

These parameters are encoded in the following TLV format when included in CASE / PASE session establishment:

```
sed-parameter-struct => STRUCTURE [ tag-order ]
{
Ê SLEEPY_IDLE_INTERVAL    [1, optional] : UNSIGNED INTEGER [ range 32-bits ],
Ê SLEEPY_ACTIVE_INTERVAL  [2, optional] : UNSIGNED INTEGER [ range 32-bits ],
}
```


Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Chapter 3. Cryptographic Primitives

This chapter introduces the various cryptographic primitives, algorithms and protocol building blocks used in this protocol. It introduces for each of them a functional abstraction that can be referred to in the other chapters of this specification. This chapter also maps those cryptographic primitives to specific instances with the corresponding appropriate informative or normative refer! ences. Wherever relevant, it also gives necessary or relevant information about the use of these mappings in a specific context to achieve a compliant implementation.

Given a version of the Message Format , the cryptographic primitives are mapped to specific instances. There is no cryptosuite negotiation in this protocol: one version of the Message Format has one cryptosuite as defined in this chapter.

Each section defines cryptographic primitives generically, together with concrete mappings to spe! cific instances of these cryptographic primitives for version 1.0 of the Message Format . This chapter can also be used as guidance about which cryptographic primitives need to be supported by a device, but it must be noted that not all devices will have to support all of them. For example, a device may not require the Crypto_PBKDF() primitive, as values based on this operation could in some instances be precomputed and stored during the manufacturing process of the device. The proposed functional mapping in this chapter is normative with respect to the values computed by the functions but informative with respect to the way the functions are interfaced within imple! mentations. For example, a function returning both a boolean to indicate success and a value if the operation is successful could also be implemented using exception mechanisms instead of return! ing a boolean.

It must also be noted that not all cryptographic primitives are exposed to the other parts of the specification. For example, the Crypto_TRNG() primitive SHALL NOT be called outside of the Crypto_! DRBG() implementation.

The cryptographic primitives discussed below operate on data local to the host. Where more com! plex data types are present and their external representation is applicable, the chapter notes the details of the encoding. Simple multi-byte data types without any additional context are assumed to be in host byte order when they are used internally to a procedure, unless otherwise stated.

All octet strings are presented with first octet having index 0, and presented from left to right for indices 0 through N-1 for an octet string of length N.

3.1. Deterministic Random Bit Generator (DRBG)

This protocol relies on random numbers for many security purposes. For example, random num! bers are used in generating secret keys, counters, cryptographic signature generation random secrets, etc. Those random numbers SHALL be generated using the Crypto_DRBG() function.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Function and description
bit[len] Crypto_DRBG(int len) Returns an array of len random bits.

Mapping (Version 1.0)
Crypto_DRBG() SHALL be implemented with one of the following DRBG algorithms as defined in NIST SP800-90A (the choice of which one is left to the manufacturer because the choice has no impact on the interoperability): ¥ CTR DRBG (with AES-CTR) ¥ HMAC DRBG (with SHA-256) ¥ HMAC DRBG (with SHA-512) ¥ Hash DRBG (with SHA-256) ¥ Hash DRBG (with SHA-512) Crypto_DRBG() SHALL be seeded using Crypto_TRNG() with at least 256 bits of entropy (see among others Chapter 4 and Section 8.4 of NIST SP800-90A).

3.2. True Random Number Generator (TRNG)

A TRNG (aka. Entropy Source) is required to provide an entropy seed as an input to the DRBG algo! rithm.

Function and description
bit[len] Crypto_TRNG(int len) Returns an array of len random bits.

Mapping (Version 1.0)
Crypto_TRNG() MAY be implemented according to the NIST SP800-90B implementation guidelines but alternate implementations MAY be used.

In accordance with good security practices, the Crypto_TRNG() SHALL never be called directly but rather SHALL be used in the implementation of Crypto_DRBG()

3.3. Hash function (Hash)

Crypto_Hash() computes the cryptographic hash of a message.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Function and description
byte[CRYPTO_HASH_LEN_IN_BYTES] Crypto_Hash(byte[] message) Returns the cryptographic hash digest of the message

Mapping (Version 1.0)
int CRYPTO_HASH_LEN_BITS := 256
int CRYPTO_HASH_LEN_BYTES := 32
int CRYPTO_HASH_BLOCK_LEN_BYTES := 64
Crypto_Hash(message) := Ê byte[CRYPTO_HASH_LEN_BYTES] SHA-256(M := message) SHA-256() SHALL be computed as defined in Section 6.2 of FIPS Ê180-4.

3.4. Keyed-Hash Message Authentication Code (HMAC)

Crypto_HMAC() computes the cryptographic keyed-hash message authentication code of a message.

Function and description
byte[CRYPTO_HASH_LEN_BYTES] Crypto_HMAC(byte[] key, byte[] message) Returns the cryptographic keyed-hash message authentication code of a message using the given key.

Mapping (Version 1.0)
Crypto_HMAC(key, message) := Ê byte[CRYPTO_HASH_LEN_BYTES] HMAC(K := key, text := message) HMAC() SHALL be computed as defined in FIPS Ê198-1 using Crypto_Hash() as the underlying hash function H (this is also referred to as HMAC-SHA256()) and CRYPTO_HASH_LEN_BYTES is defined in Section 3.3, ÒHash function (Hash)Ó.

3.5. Public Key Cryptography

Matter specifies the following scheme and parameters for public key cryptography.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

3.5.1. Group

The public key cryptography of Matter relies on the group defined in the following mapping table.

Mapping (Version 1.0)
Matter public key cryptography SHALL be based on Elliptic Curve Cryptography (ECC) with the elliptic curve: secp256r1 defined in Section 2.4.2 of SECÊ2 (Informative: Note that this curve is also referred to as NIST P-256 or prime256v1 in FIPSÊ186-4 and NISTÊ800-186)
PrivateKey is an opaque data type to hold either the private key or any handle or reference that allows other primitives to access the corresponding private key.
PublicKey is an opaque data type to hold the public key or any handle or reference that allows other primitives to access the corresponding public key. A public key is a point on the elliptic curve. (Note: at places in the specification where public keys are to be explicitly transmitted, the format in which they are transmitted is specified.)
int CRYPTO_GROUP_SIZE_BITS := 256
int CRYPTO_GROUP_SIZE_BYTES := 32
int CRYPTO_PUBLIC_KEY_SIZE_BYTES : = (2 * CRYPTO_GROUP_SIZE_BYTES) + 1 is the size in bytes of the public key representation when encoded using the uncompressed public key format as specified in section 2.3 of SECÊ1.
struct { Ê PublicKey publicKey; Ê PrivateKey privateKey; } KeyPair;

3.5.2. Key generation

Crypto_GenerateKeyPair() is the function to generate a key pair.

Function and description
KeyPair Crypto_GenerateKeyPair() Generates a key pair and returns a KeyPair.

Mapping (Version 1.0)
Crypto_GenerateKeypair() := Ê KeyPair ECCGenerateKeypair() ECCGenerateKeypair()SHALL generate a key pair according to Section 3.2.1 of SECÊ1.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

3.5.3. Signature and verification

Crypto_Sign() is used to sign a message, and Crypto_Verify() is used to verify a signature on a message.

These functions either generate or verify a signature of type Signature defined by the following mapping.

Mapping (Version 1.0)

```
struct {
Ê  byte[CRYPTO_GROUP_SIZE_BYTES] r,
Ê  byte[CRYPTO_GROUP_SIZE_BYTES] s
} Signature
```

3.5.3.1. Signature

Function and description

```
Signature
Crypto_Sign(
Ê  PrivateKey privateKey,
Ê  byte[] message)
```

Returns the signature of the message using the privateKey .

Mapping (Version 1.0)

```
Crypto_Sign(privateKey, message) :=
Ê  Signature ECDSASign(dU := privateKey, M := message)
```

ECDSASign() SHALL be the ECDSA signature function as defined in Section 4.1 of SECÊ1 using Crypto_Hash() as the underlying hash Hash() function.

3.5.3.2. Signature verification

Function and description

```
boolean
Crypto_Verify(
Ê  PublicKey publicKey,
Ê  byte[] message,
Ê  Signature signature)
```

Verifies the signature of the message using the publicKey , returns TRUE if the verification succeeds, FALSE otherwise.

csa connectivity standards alliance Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Mapping (Version 1.0)

```
Crypto_Verify(publicKey, message, signature) :=
Ê  boolean ECDSAVerify(QU := publicKey, M := message, S := signature)
```

ECDSAVerify() SHALL be the ECDSA signature verification function as defined in Section 4.1.4 of SECÊ1 using Crypto_Hash() as the underlying hash function Hash(); returns TRUE if the verification succeeds and FALSE otherwise.

3.5.4. ECDH

Crypto_ECDH() is used to compute a shared secret from the Elliptic Curve Diffie-Hellman (ECDH) protocol.

Function and description

```
byte[CRYPTO_GROUP_SIZE_BYTES]
Crypto_ECDH(
Ê  PrivateKey myPrivateKey,
Ê  PublicKey theirPublicKey)
```

Computes a shared secret using Elliptic Curve Diffie-Hellman.

Mapping (Version 1.0)

```
Crypto_ECDH(myPrivateKey, theirPublicKey) :=
Ê  byte[CRYPTO_GROUP_SIZE_BYTES] ECDH(dU := myPrivateKey, QV := theirPublicKey)
```

The output of ECDH() SHALL be the serialization of the x-coordinate of the resultant point as defined in Section 3.3.1 of SECÊ1

3.5.5. Certificate validation

Crypto_VerifyChain() is used to verify Matter certificates .

Crypto_VerifyChainDER() is used to verify public key X.509 v3 certificates in X.509 v3 DER format.

Function and description

```
boolean
Crypto_VerifyChain(MatterCertificate[] certificates)
```

Given Matter certificates , Crypto_VerifyChain() performs all the necessary validity checks on cer!tificates , taking in account that the notion of "current time" for the purposes of validation SHALL abide by the rules in Section 3.5.6, ÒTime and date considerations for certificate path validationÓ .

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Function and description
boolean Crypto_VerifyChainDER(DERCertificate[] certificates) Given a list of DER-encoded certificates in RFC5280 format, starting at the end-entity (leaf) certificate, and following the chain of trust towards the root, Crypto_VerifyChainDER() performs all the necessary validity checks on certificates . The Validity period validation for the root and optional intermediate certificates is performed against the notBefore timestamp of the end-entity (leaf certificate) used as value for the current time.

Mapping (Version 1.0)
Crypto_VerifyChain(certificates) := Ê boolean verified verified is TRUE if the Matter certificates are verified as prescribed by RFC5280.
Crypto_VerifyChainDER(certificates) := Ê boolean verified verified is TRUE if the certificates are verified as prescribed by RFC5280.

The primitives as described above verify cryptographic integrity of the certificate chains. This specification imposes a number of additional constraints on certificates discussed below in sections on Device Attestation Certificates , Node Operational Certificates and Certificate Common Conventions .

3.5.6. Time and date considerations for certificate path validation

The Basic Path Validation algorithm in RFC5280 mandates the consideration of the "current time" against the validity period (notBefore , notAfter fields) when validating paths. The usage of "current time" assumes that such a time is available and correct, which is a strong assumption when considering some constrained devices or devices only locally reachable on a network in the absence of infrastructure to synchronize time against a global real-time reference.

When the Crypto_VerifyChain primitive is used, rather than overriding the Basic Path Validation algorithm of RFC5280, Nodes SHALL consider the following definition of "current time" that accounts for the possible lack of a real time reference:

- ¥ If a Node has a current real-time clock value which is trusted according to implementation-defined means to be accurate with regard to global real-time, whether using Time Synchronization features of this specification or other means, then it SHALL use that time;
- ¥ Otherwise, the current time SHALL be set to the last-known-good UTC time.

Upon failure to validate a certificate path, where the only reason for failure is an invalid validity period of a path element, a Node MAY apply a policy of its choice to determine whether to ignore

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

this failure and still consider the path valid.

3.5.6.1. Last Known Good UTC Time

Nodes SHALL maintain a stored Last Known Good UTC Time. This time is used as a fallback for cryptographic credentials expiry enforcement, if all other available time synchronization mecha! nisms fail.

The last known good UTC time SHALL be updated at commissioning and MAY be updated after a successful time synchronization, or by an embedded time in an OTA. Nodes SHOULD store a Last Known Good UTC Time value to persistent storage at least once a month. A NodeÕs initial out-of-box Last Known Good UTC time SHALL be the compile-time of the firmware.

A Node MAY adjust the Last Known Good UTC Time backwards if it believes the current Last Known Good UTC Time is incorrect and it has a good time value from a trusted source. The Node SHOULD NOT adjust the Last Known Good UTC to a time before the later of:

- ¥ The build timestamp of its currently running software image
- ¥ The not-before timestamp of any of its operational certificates (see Section 6.4.5, ÒNode Opera! tional Credentials CertificatesÓ).

If a Node has used the Last Known Good UTC Time, it SHOULD recheck its security materials and existing connections if it later achieves time synchronization.

3.6. Data Confidentiality and Integrity

Symmetric block ciphers are used to provide message security.

All unicast and multicast messages between Nodes requiring protection for confidentiality and integrity with data origin authentication SHALL use Authenticated Encryption with Associated Data (AEAD) as primitive to protect those messages.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Mapping (Version 1.0)
Data confidentiality and integrity SHALL use the AES-CCM mode as defined in NISTÊ800-38C with the following parameters: ¥ int CRYPTO_SYMMETRIC_KEY_LENGTH_BITS := 128 (this is the key length of the underlying block cipher in bits) ¥ int CRYPTO_SYMMETRIC_KEY_LENGTH_BYTES := 16 (this is the key length of the underlying block cipher in bytes) ¥ int CRYPTO_AEAD_MIC_LENGTH_BITS := 128 (this is the MIC length in bits) ¥ int CRYPTO_AEAD_MIC_LENGTH_BYTES := 16 (this is the MIC length in bytes) ¥ int CRYPTO_AEAD_NONCE_LENGTH_BYTES := 13 ¥ Key length SHALL be CRYPTO_SYMMETRIC_KEY_LENGTH_BITS bits. ¥ MIC length SHALL be CRYPTO_AEAD_MIC_LENGTH_BITS bits. ¥ The parameter q SHALL be 2 (length of encoding of maximum length) as specified in Appendix A.1 of NISTÊ800-38C. ¥ The parameter n SHALL be CRYPTO_AEAD_NONCE_LENGTH_BYTES (length of nonce in bytes) as specified in Appendix A.1 of NISTÊ800-38C.
SymmetricKey is an opaque data type to hold a symmetric block cipher key or any handle or reference that allows other primitives to access the corresponding key.

3.6.1. Generate and encrypt

Function and description
byte[lengthInBytes(P) + CRYPTO_AEAD_MIC_LENGTH_BYTES] Crypto_AEAD_GenerateEncrypt(Ê SymmetricKey K, Ê byte[lengthInBytes(P)] P, Ê byte[] A, Ê byte[CRYPTO_AEAD_NONCE_LENGTH_BYTES] N)
Performs the generate and encrypt computation on payload P and the associated data A using the key K and a nonce N; the output contains the ciphertext and the tag truncated to Tlen bits (the encoding of the output depends on the mapping to the specific instance of the cryptographic primitive).

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Mapping (Version 1.0)

```
Crypto_AEAD_GenerateEncrypt(K, P, A, N) :=
Ê   byte[lengthInBytes(P) + CRYPTO_AEAD_MIC_LENGTH_BYTES] AES-CCM-GenerateEncrypt(K
:= K, P := P, A := A, N := N, Tlen := CRYPTO_AEAD_MIC_LENGTH_BITS)
```

AES-CCM-GenerateEncrypt() SHALL be the function described in Section 6.1 of NISTÊ800-38C with the counter generation function of Appendix A.3 of NISTÊ800-38C and the formatting function as defined in Appendix A.2 of NISTÊ800-38C, returns the encoding of the ciphertext and the tag of length Tlen bits, as specified in Section 6.1 of NISTÊ800-38C as a byte array.

3.6.2. Decrypt and verify

Function and description

```
{boolean success, byte[lengthInBytes(P)] payload}
Crypto_AEAD_DecryptVerify(
Ê   SymmetricKey K,
Ê   byte[lengthInBytes(P) + CRYPTO_AEAD_MIC_LENGTH_BYTES] C,
Ê   byte[] A,
Ê   byte[CRYPTO_AEAD_NONCE_LENGTH_BYTES] N)
```

Performs the decrypt and verify computation on the combined ciphertext and tag C and the associ! ated data A using the key K and a nonce N. Note that the encoding of C depends on the mapping of the specific instance of the cryptography primitive.

This function has two outcomes:

- ¥ If tag verification succeeds, the success output is TRUE and the payload array contains the decrypted payload P.
- ¥ If tag verification fails, the success output is FALSE and the contents of the payload array is undefined.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Mapping (Version 1.0)

```
Crypto_AEAD_DecryptVerify(K, C, A, N) :=
Ê   {boolean, byte[lengthInBytes(P)]} AES-CCM-DecryptVerify(K := K, C := C, A := A,
N := N, Tlen := CRYPTO_AEAD_MIC_LENGTH_BITS)
```

AES-CCM-DecryptVerify() SHALL be the function described in Section 6.2 of NISTÊ800-38Cwith the counter generation function of Appendix A.3 of NISTÊ800-38Cand the formatting function as defined in Appendix A.2 of NISTÊ800-38Cand C SHALL be a byte array containing the ciphertext as specified in Section 6.2 of NISTÊ800-38C.

- ¥ If tag verification succeeds, the success output is TRUEand the payload array contains the decrypted payload P.
- ¥ If tag verification fails, the success output is FALSEand the contents of the payload array is undefined.

3.7. Message privacy

Message privacy is implemented using a block cipher in CTR mode.

Mapping (Version 1.0)

Message privacy SHALL use the AES-CTR mode as defined in NISTÊ800-38Awith the following parameters:

- ¥ int CRYPTO_PRIVACY_NONCE_LENGTH_BYTES := 13
- ¥ Key length SHALL be CRYPTO_SYMMETRIC_KEY_LENGTH_BITS bits.

3.7.1. Privacy encryption

Function and description

```
byte[lengthInBytes(M)]
Crypto_Privacy_Encrypt(
Ê   SymmetricKey K,
Ê   byte[lengthInBytes(M)] M,
Ê   byte[CRYPTO_PRIVACY_NONCE_LENGTH_BYTES] N)
```

Performs the encryption of the message Musing the key K and a nonce N; the output contains the data Mencrypted.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Mapping (Version 1.0)

```
Crypto_Privacy_Encrypt(K, M, N) :=
Ê   byte[lengthInBytes(M)]
Ê   AES-CTR-Encrypt(K := K, P := M, N := N)
```

AES-CTR-Encrypt() SHALL be the encryption function described in Section 6.5 of NISTÊ800-38A with the sequence of counters T being generated according to the counter generation function of Appendix A.3 of NISTÊ800-38C using N and the value of q = 2 ; returns the encrypted message as a byte array.

3.7.2. Privacy decryption

Function and description

```
byte[lengthInBytes(C)]
Crypto_Privacy_Decrypt(
Ê   SymmetricKey K,
Ê   byte[lengthInBytes(C)] C,
Ê   byte[CRYPTO_PRIVACY_NONCE_LENGTH_BYTES] N)
```

Performs the decryption of C using the key K and a nonce N; the output M is the decryption of C

Mapping (Version 1.0)

```
Crypto_Privacy_Decrypt(K, C, N) :=
Ê   byte[lengthInBytes(C)]
Ê   AES-CTR-Decrypt(K := K, C := C, N := N)
```

AES-CTR-Decrypt() SHALL be the decryption function described in Section 6.5 of NISTÊ800-38A with the sequence of counters T being generated according to the counter generation function of Appendix A.3 of NISTÊ800-38C using N and the value of q = 2 ; returns the decrypted message as a byte array.

3.8. Key Derivation Function (KDF)

Matter specifies the following key derivation function to generate encryption keys.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Function and description

```
bit[len]
Crypto_KDF(
Ê    byte[] inputKey,
Ê    byte[] salt,
Ê    byte[] info,
Ê    int len)
```

Returns the key of len bits derived from inputKey using the salt and the info ; len SHALL be a mul! tiple of 8.

Mapping (Version 1.0)

```
Crypto_KDF(inputKey, salt, info, len) :=
Ê    bit[len] KDM(Z := inputKey, OtherInput := {salt := salt, L := len, FixedInfo :=
info})
```

KDM()SHALL be the HMAC-based KDF function with Crypto_HMAC(key := salt, message := x) as the auxiliary function H as defined in Section 4.1 Option 2 of NISTÊ800-56C; it returns a bit array of len bits.

When multiple keys of the same length are generated by a single KDF call, the following shorthand notation can be used:

```
Key1 || Key2 || Key3 = Crypto_KDF
Ê   (
Ê    inputKey = inputKeyMaterial,
Ê    salt = [],
Ê    info = [],
Ê    // 3 below matches number of keys expressed in concatenated output
Ê    len = 3 * CRYPTO_SYMMETRIC_KEY_LENGTH_BITS
Ê   )
```

This is equivalent to the following:

```
Keys = Crypto_KDF
Ê   (
Ê    inputKey = inputKeyMaterial,
Ê    salt = [],
Ê    info = [],
Ê    len = 3 * CRYPTO_SYMMETRIC_KEY_LENGTH_BITS
Ê   )
```

1. Set Key1to the CRYPTO_SYMMETRIC_KEY_LENGTH_BITS most significant bits of Keys.
2. Set Key2to the next CRYPTO_SYMMETRIC_KEY_LENGTH_BITS significant bits of Keys.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

3. Set Key3 to the CRYPTO_SYMMETRIC_KEY_LENGTH_BITS least significant bits of Keys

3.9. Password-Based Key Derivation Function (PBKDF)

Matter specifies the following password-based key derivation function to compute a derived key from a cryptographically weak password.

Function and description

```
bit[len]
Crypto_PBKDF(
Ê   byte[] input,
Ê   byte[] salt,
Ê   int iterations,
Ê   int len)
```

Returns a value of len bits derived from the input using the salt and iterations iterations.

Type and description

```
STRUCTURE Crypto_PBKDFParameterSet
```

Maintains the set of parameters exchanged between a Commissioner and a Commissionee during their pairing.

Mapping (Version 1.0)
int CRYPTO_PBKDF_ITERATIONS_MIN := 1000
int CRYPTO_PBKDF_ITERATIONS_MAX := 100000

```
Crypto_PBKDF(input, salt, iterations, len) :=
Ê   bit[len] PBKDF2(P := input, S := salt, C := iterations, kLen := len)
```

PBKDF2() SHALL be the HMAC-based PBKDF function with Crypto_HMAC(key := P, message := U[j-1]) as the auxiliary function HMAC as defined in Section 5.3 of NISTÊ800-132, it returns a bit array of len bits.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Mapping (Version 1.0)

```
Crypto_PBKDFParameterSet => STRUCTURE [ tag-order ]
{
Ê   iterations [1] : UNSIGNED INTEGER [ range 32-bits ],
Ê   salt [2] : OCTET STRING [ length 16..32 ],
}
```

¥ iterations : An integer value specifying the number of PBKDF2 iterations: CRYPTO_PBKDF_ITERATIONS_MIN <= iterations <= CRYPTO_PBKDF_ITERATIONS_MAX.

¥ salt : A random value per device of at least 16 bytes and at most 32 bytes used as the PBKDF2 salt.

3.10. Password-Authenticated Key Exchange (PAKE)

This protocol uses password-authenticated key exchange (PAKE) for the PASE protocol.

Mapping (Version 1.0)

Matter uses SPAKE2+ as described in SPAKE2+ as PAKE with:

¥ The SPAKE2+ verifier is the Commissionee/Responder and the SPAKE2+ prover is the Commissioner/Initiator

¥ Crypto_PBKDF() as underlying PBKDF (see Section 3.9, ÒPassword-Based Key Derivation Function (PBKDF)Ó), with arguments as described in the definition of Crypto_PAKEValues_Initiator

¥ NIST P-256 elliptic curve as underlying group (see Section 3.5.1, ÒGroupÓ).

%SPAKE2+ requires two additional points on the curve: M and N. The values of M and N are taken from the draft version 2 of the SPAKE2+ specification (SPAKE2+) and are listed below in compressed format (format defined in section 2.3 of SECÊ1)

& M = 02886e2f97ace46e55ba9dd7242579f2993b64e16ef3dcab95afd497333d8fa12f

& N = 03d8bbd6c639c62937b04d997f38c3770719c629d7014d49a24b4f98baa1292b49

¥ Crypto_Hash() as underlying hash function (see Section 3.3, ÒHash function (Hash)Ó).

¥ Crypto_HMAC() as underlying HMAC function (see Section 3.4, ÒKeyed-Hash Message Authentication Code (HMAC)Ó).

¥ KDF(info, key, salt) := Crypto_KDF(key, salt, info, CRYPTO_HASH_LEN_BITS) as underlying KDF function (see Section 3.8, ÒKey Derivation Function (KDF)Ó).

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Mapping (Version 1.0)

Crypto_PAKEValues_Initiator := (w0, w1) where w0and w1SHALL be computed as follows:

```
CRYPTO_W_SIZE_BYTES := CRYPTO_GROUP_SIZE_BYTES + 8
CRYPTO_W_SIZE_BITS := CRYPTO_W_SIZE_BYTES * 8

byte w0s[CRYPTO_W_SIZE_BYTES] || byte w1s[CRYPTO_W_SIZE_BYTES] =
Ê    (byte[2 * CRYPTO_W_SIZE_BYTES])
Ê    bit[2 * CRYPTO_W_SIZE_BITS]
Ê    Crypto_PBKDF(passcode, salt, iterations, 2 * CRYPTO_W_SIZE_BITS)
byte w0[CRYPTO_GROUP_SIZE_BYTES] = w0s mod p
byte w1[CRYPTO_GROUP_SIZE_BYTES] = w1s mod p
```

where:

- ¥ modis the mathematical modulo operation and || is the string concatenation or split operator.
- ¥ passcode, is the Passcode defined in Section 5.1.1.6, ÒPasscodeÓ, serialized as little-endian over 4 octets. For example, passcode 18924017would be encoded as the octet string f1:c1:20:01 and the passcode 00000005would be encoded as the octet string 05:00:00:00 .
- ¥ p is the order of the underlying elliptic curve.
- ¥ Both w0sand w1sSHALL have a length equal to (CRYPTO_GROUP_SIZE_BYTES + 8)
- ¥ salt and iterations are extracted from the Crypto_PBKDFParameterSetvalues.
- ¥ The pair (w0,w1) is also referred to as Commissioner PAKE input

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Mapping (Version 1.0)

Crypto_PAKEValues_Responder := (w0, L) where w0 and L SHALL be computed as follows:

```
byte w0s[CRYPTO_W_SIZE_BYTES] || byte w1s[CRYPTO_W_SIZE_BYTES] =
Ê   (byte[2 * CRYPTO_W_SIZE_BYTES])
Ê   bit[2 * CRYPTO_W_SIZE_BITS]
Ê   Crypto_PBKDF(passcode, salt, iterations, 2 * CRYPTO_W_SIZE_BITS)
byte w0[CRYPTO_GROUP_SIZE_BYTES] = w0s mod p
byte w1[CRYPTO_GROUP_SIZE_BYTES] = w1s mod p
byte L[CRYPTO_PUBLIC_KEY_SIZE_BYTES] = w1 * P
```

where:

- ¥ passcode is the Passcode defined in Section 5.1.1.6, ÒPasscodeÓ.
- ¥ p is the order of the elliptic curve to be used.
- ¥ Both w0s and w1s SHALL have a length equal to (CRYPTO_GROUP_SIZE_BYTES + 8)
- ¥ salt and iterations are extracted from the Crypto_PBKDFParameterSet values.
- ¥ P is the generator of the underlying elliptic curve.
- ¥ The computation of Crypto_PAKEValues_Responder SHALL be computed at device manufacturing time and w0 and L SHALL be stored in the Responder and w1 SHALL NOT be stored in the Responder.
- ¥ The pair (w0,L) is also referred to as Commissionee PAKE input or verification value

3.10.1. Computation of pA

Mapping (Version 1.0)

```
Crypto_pA(Crypto_PAKEValues_Initiator) :=
Ê   byte pA[CRYPTO_PUBLIC_KEY_SIZE_BYTES]
```

pA is in uncompressed public key format as specified in section 2.3 of SECÊ1 pA SHALL be com! puted as specified in SPAKE2+.

3.10.2. Computation of pB

Mapping (Version 1.0)

```
Crypto_pB(Crypto_PAKEValues_Responder) :=
Ê   byte pB[CRYPTO_PUBLIC_KEY_SIZE_BYTES]
```

pB is in uncompressed public key format as specified in section 2.3 of SECÊ1 pB SHALL be com! puted as specified in SPAKE2+.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

3.10.3. Computation of transcript TT

Mapping (Version 1.0)

```
Crypto_Transcript(PBKDFParamRequest, PBKDFParamResponse, pA, pB) :=
Ê    byte[] TT
```

Crypto_Transcript() SHALL compute TT as specified in SPAKE2+ with:

```
Context := Crypto_Hash("Matter PAKE V1 Commissioning" || PBKDFParamRequest ||
PBKDFParamResponse)

TT :=
Ê    lengthInBytes(Context)  || Context           ||
Ê    0x0000000000000000      || 0x0000000000000000 ||
Ê    lengthInBytes(M)        || M                 ||
Ê    lengthInBytes(N)        || N                 ||
Ê    lengthInBytes(pA)       || pA                ||
Ê    lengthInBytes(pB)       || pB                ||
Ê    lengthInBytes(Z)        || Z                 ||
Ê    lengthInBytes(V)        || V                 ||
Ê    lengthInBytes(w0)       || w0
```

Z and V SHALL be computed from pA and pB as specified in SPAKE2+.

Note the two 0x0000000000000000 null-lengths indicate that no identities are present and each null-lengths is 8 bytes wide since it is specified by the SPAKE2+ specification that lengths are eight-byte little-endian numbers. The SPAKE2+ specification indicates that we must include these length fields.

Note in case PBKDFParamRequest and PBKDFParamResponse messages are not exchanged, they SHALL be replaced by empty strings in the Context computation.

3.10.4. Computation of cA, cB and Ke

Mapping (Version 1.0)

```
Crypto_P2(TT, pA, pB) :=
Ê    {byte cA[CRYPTO_HASH_LEN_BYTES],
Ê     byte cB[CRYPTO_HASH_LEN_BYTES],
Ê     byte Ke[CRYPTO_HASH_LEN_BYTES/2]}
```

Crypto_P2() SHALL compute cA, cB and Ke as specified in SPAKE2+ with cA := CRYPTO_HMAC(KcA,pB) and cB := CRYPTO_HMAC(KcB,pA)

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Chapter 4. Secure Channel

4.1. General Description

The Secure Channel and Message Layer provides a consistent networking service substrate to allow Nodes to communicate securely with one another.

During commissioning and unicast communication, a discovery mechanism is provided to deter! mine peer IPv6 addresses and operational parameters. Secure session establishment mechanisms are provided using either certificates (CASE) or shared passcodes (PASE).

4.1.1. Messages

Communication is performed using messages. Messages are either secured or unsecured.

Each message has a Session Type and Session ID in order to identify whether it is secured and how it is to be decrypted and authenticated if it is. Each message has a Message Counter field in order to uniquely identify the message for the purposes of security and duplicate detection.

Operational communication is defined as traffic that uses the secured Matter message format between commissioned nodes over an IP transport. All operational communication has message security enabled. Operational communication between Nodes can be either unicast or multicast.

Unsecured communication is strictly limited to:

- Discovery , which does not use the Matter message format.
- User Directed Commissioning (UDC) , which uses unsecured messages to initiate the commission! ing process.
- Session establishment , which uses unsecured messages to establish a CASE or PASE session.

4.1.1.1. Message Types

Messages are defined as either a control message or data message. Most messages are data mes! sages. Control messages are reserved for internal protocols such as MCSP to initialize security. Both message types are identical in format, but use separate message counter domains so they can oper! ate securely over the same security key.

4.1.1.2. Message Transports

Messages are of finite size and are sent as individual packets over the supported transports:

- UDP transports each message as a separate datagram. Messages support a basic reliability pro! tocol called MRP for use when the underlying transport (in this case UDP) doesnÕt provide such features.
- TCP transports each message with a length prepended, performing segmentation and reassem! bly as needed.
- BTP transports each message over BLE as a separate SDU, performing segmentation and

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

reassembly as needed.

BTP is provided as a transport protocol for commissioning. TCP and MRP (UDP with added reliability) are provided as transport protocols for operational messaging.

4.1.1.3. Message Exchanges

Messages provide an Exchange Layer to track related messages that make up small, discrete transactions. The Exchange Layer provides this transaction tracking facility to the Interaction Model Layer above, providing a means to multiplex multiple such concurrent transactions over a given underlying session. The Exchange Layer also integrates the Message Reliability Protocol (MRP) as a service for use over UDP transports. This Message Layer architecture is shown below in Figure 6, ÒMessage Layer StackÓ.

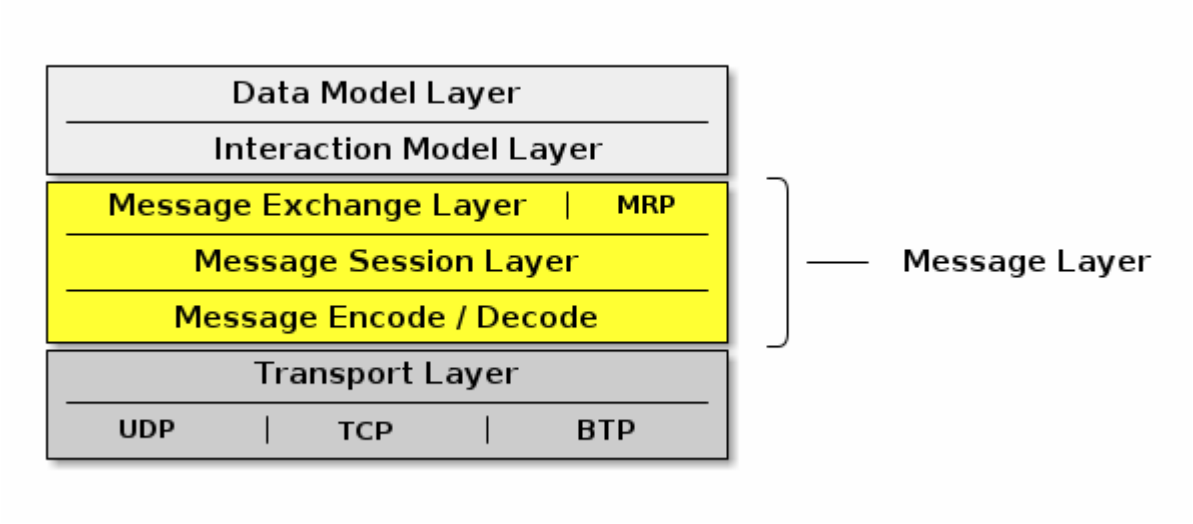


Figure 6. Message Layer Stack

4.2. IPv6 Reachability

This section describes IPv6 network configuration requirements to enable IPv6 reachability between Nodes. As described in Section 2.3, ÒNetwork TopologyÓ, a Matter network may be composed of one or more IPv6 networks.

In a single network configuration, all Matter Nodes are attached to the same IPv6 link. A single network configuration may consist of a single bridged Wi-Fi / Ethernet network where all nodes attached to that network are part of the same broadcast domain. When all Matter Nodes are attached to the same Wi-Fi / Ethernet network, link-local IPv6 addressing is sufficient - no additional IPv6 network infrastructure is required.

In a multiple network configuration, a Matter network is composed of (typically one) infrastructure network and one or more stub networks. Unlike an infrastructure network, stub networks do not serve as transit networks. Typically, the infrastructure network is a bridged Wi-Fi / Ethernet network and the Thread networks are stub networks. A stub router connects a stub network to an infrastructure network and provides IPv6 reachability between the two networks.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

4.2.1. Stub Router Behavior

A stub router SHALL implement [draft-lemon-stub-networks]. In a multiple network configuration, both the infrastructure and stub networks require routable IPv6 addresses to communicate across networks. A routable IPv6 address SHALL have global scope (e.g. GUA or ULA) [RFCÊ4007] and SHALL be constructed out of a prefix advertised as on-link. If there is no routable prefix on a given network, the stub router SHALL provide its own routable prefix. Note that ThreadÕs "on-mesh pre! fix" is equivalent to Wi-Fi / EthernetÕs "on-link prefix".

Stub routers SHALL advertise reachability to all routable prefixes on the adjacent network. A stub router connecting a Thread network SHALL advertise reachability to all of the Thread networkÕs routable prefixes to the adjacent infrastructure network using Route Information Options [RFCÊ4191] contained in Router Advertisements [RFCÊ4861]. That same stub router SHALL also advertise reachability to all of the infrastructure networkÕs routable prefixes to the adjacent Thread network in the Thread Network Data [Thread specification].

4.2.2. Matter Node Behavior

Matter Nodes SHALL configure a link-local IPv6 address. In addition, Nodes SHALL support configu! ration of at least three routable IPv6 addresses (in addition to the link-local and, in the case of Thread, mesh-local addresses). On a Wi-Fi / Ethernet interface, ICMPv6 Router Advertisement (RA) messages advertise prefixes for use on the link [RFCÊ4861]. On a Thread interface, the Thread Net! work Data contains prefixes for use on the link [Thread specification]. If a Node receives an on-link prefix that allows autonomous configuration on a given interface and the Node has fewer than three routable IPv6 addresses configured, the Node SHALL autonomously configure an IPv6 address out of that prefix.

Matter Nodes SHALL also configure routes to adjacent networks. On Wi-Fi / Ethernet networks, Nodes SHALL process Route Information Options [RFCÊ4191] and configure routes to IPv6 prefixes assigned to stub networks via stub routers. Wi-Fi / Ethernet interfaces SHALL support maintaining at least 16 different routes configured using Route Information Options. On Thread networks, Nodes SHALL route according to routing information provided in the Thread Network Data [Thread specification]. Thread devices SHALL support as many routes as can be encoded in the Thread Net! work Data.

Matter Nodes SHALL support a number of IPv6 neighbor cache entries at least as large as the num! ber of supported CASE sessions plus the number of supported routes.

4.3. Discovery

This section describes Service Advertising and Discovery for Matter.

Service Advertising and Discovery is used within Matter in the following contexts:

- ¥ Commissionable Node Discovery
- ¥ Operational Discovery
- ¥ Commissioner Discovery
- ¥ User Directed Commissioning

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Service Advertising and Discovery for Matter uses IETF Standard DNS-Based Service Discovery (DNS'SD) [RFC6763]. Matter requires no modifications to IETF Standard DNS'SD.

Using DNS'SD means that both the unicast IPv6 address and port of the offered service are discovered, freeing Matter from requiring a single preallocated fixed port. This also makes it possible to run multiple instances of Matter software on a single device, because each instance has its own dynamically allocated port, instead of conflicting over attempting to use the same preallocated fixed port.

On Wi'Fi and Ethernet networks today, DNS'SD [RFC6763] uses Multicast DNS [RFC6762] for zero-configuration operation.

Since Matter protocols must support IPv6 at a minimum, Matter software discovering other Matter instances SHALL process DNS AAAA (IPv6 address) records, but also MAY process DNS A (IPv4 address) records.

Because of this, where feasible in the underlying service discovery API, Matter software advertising the availability of a service SHOULD indicate that announcements and answers for this service need include only IPv6 address records, not IPv4 address records. On a general-purpose dual-stack host that supports both IPv4 and IPv6, this can be achieved by having Matter-related SRV records reference a Matter-specific target hostname that has only IPv6 address records attached. This allows a general-purpose dual-stack host to offer discoverable IPv4 addresses for legacy client software that still requires IPv4, while offering optimized IPv6-only address discovery for Matter purposes.

Similarly, since Matter does not use IPv4, Matter software discovering other Matter instances SHOULD NOT expect any IPv4 addresses included in responses.

These two items address the content of service discovery messages. When using Multicast DNS similar efficiency questions arise related to the delivery of those service discovery messages, sent over IPv4, IPv6, or both.

Where supported in the underlying service discovery API, Matter software using Multicast DNS to advertise the availability of a service SHOULD indicate that announcements and answers for this service need only be performed over IPv6.

Similarly, where supported in the underlying service discovery API, Matter application software using Multicast DNS to issue service discovery queries SHOULD indicate that these queries need only be performed over IPv6.

These optimizations reduce both the size and the number of multicast packets, which is particularly beneficial on Wi'Fi networks. A Matter device that only supports IPv6 gets these optimizations automatically, simply by virtue of not supporting IPv4 at all.

For Thread mesh networks, where excessive use of multicast would be detrimental [RFC7553], DNS'SD uses Unicast DNS instead, leveraging capabilities of the Thread Service Registry on the Thread Border Router [draft-lemon-stub-networks].

Conceptually, the DNS'SD [RFC6763] information being communicated is identical to when Multicast DNS [RFC6762] is being used, except that the information is communicated in unicast packets

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

to and from a designated Service Registry, rather than being communicated in multicast packets to and from every other node in the same broadcast domain.

Using Service Registration Protocol [SRP] and an Advertising Proxy [AdProx] running on the Thread Border Router, Matter Nodes on a Thread mesh can be discovered by other Matter Nodes on an adjacent Ethernet or Wi'Fi link, without the cost of using multicast on the Thread mesh. All Thread-connected Matter Nodes SHALL implement Service Registration Protocol.

Thread Border Routers advertise available SRP servers in the Thread Network Data [Thread specifi! cation]. Thread devices SHALL register their services using an available SRP server [Thread specifi! cation].

When Matter Nodes issue short-lived requests to other Matter Nodes, the response is sent back to the source IPv6 address and port of the request. When Matter Nodes issue long-lived requests to other Matter Nodes, by the time the response is generated the requester may have changed IPv6 address or port, so the responder may have to discover the current IPv6 address and port of the ini! tiator in order to send the response.

A Thread Border Router SHALL implement DNS'SD Discovery Proxy [RFCÊ8765] to enable clients on the Thread mesh (e.g., other Nodes) to discover services (e.g., Matter Nodes) advertised using Multi! cast DNS on an adjacent Ethernet or Wi'Fi link, also without the cost of using multicast on the Thread mesh [draft-lemon-stub-networks]. For short-lived instantaneous queries, these queries can be performed using unicast DNS over UDP to the DNS'SD Discovery Proxy. For long-lived queries with ongoing change notification, DNS Push Notifications [RFCÊ8765] with DNS Stateful Operations [RFCÊ8490] allows clients on the Thread mesh to be notified of changes to the set of discovered ser! vices without expensive polling.

In principle, the Thread mesh Service Registry can be run on any capable Node(s) within (or even outside) the Thread mesh, though in practice the Thread Border Router is an attractive candidate to offer the Service Registry. Thread Border Router devices are typically AC-powered, and typically have more capable CPUs with greater flash storage and RAM than more constrained battery-pow! ered Thread Nodes. Matter devices on Thread are dependent on Thread providing reliable service for those Thread devices on the Thread mesh. This is similar to how Matter devices on Wi'Fi are dependent on the Wi'Fi access point (AP) providing reliable service for those Wi'Fi devices using that Wi'Fi access point.

4.3.1. Commissionable Node Discovery

The Matter protocol family supports UDP and TCP for Matter commissioning of Commissionees already on the customerÕs IP network (such as Ethernet-connected Nodes, or Wi'Fi Nodes already associated to the Wi'Fi network via other means), and for the commissioning of Commissionees in conjunction with Wi'Fi Soft-AP (for Wi'Fi Nodes not already on the customerÕs IP network, when the Node does not support Matter commissioning using BLE).

For these Commissionees, Matter Commissionable Node Discovery is performed using IETF Stan! dard DNS-Based Service Discovery (DNS'SD) [RFCÊ6763] as described below.

For Matter Commissionable Node Discovery in the already-on-network and Soft-AP cases, the DNS'SD instance name SHALL be a dynamic, pseudo-randomly selected, 64-bit temporary unique

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

identifier, expressed as a fixed-length sixteen-character hexadecimal string, encoded as ASCII (UTF-8) text using capital letters, e.g., DD200C20D25AE5A7. A new instance name SHALL be selected when the Node boots. A new instance name SHALL be selected whenever the Node enters Commissioning mode. A new instance name MAY be selected at other times, as long as the instance name does not change while the Node is in commissioning mode.

When a Node receives either the OpenCommissioningWindow or the OpenBasicCommissioning!Window command, the Node SHALL only beacon on the IP network using the relevant DNS-SD properties described below.

The Matter Commissionable Node Discovery DNS'SD instance name SHALL be unique within the namespace of the local network (the .local link-local namespace of the Ethernet and Wi'Fi links [RFCÊ6762], or the unicast domain selected by the Thread Border Router for devices on the Thread mesh).

In the rare event of a collision in the selection of the 64-bit temporary unique identifier, the existing DNS'SD name conflict detection mechanism will detect this collision, and a new pseudo-randomly selected 64-bit temporary unique identifier SHALL be generated by the Matter Commissionee that is preparing for commissioning. Name conflict detection is described in Section 9 ("Conflict Resolu!tion") of the Multicast DNS specification [RFCÊ6762] and Section 2.4.3.1 ("Validation of Adds") of the Service Registration Protocol specification [SRP].

The DNS'SD service type [RFCÊ6335] for Matter Commissionable Node Discovery is _matterc._udp.

For link-local Multicast DNS the service domain SHALL be local. For Unicast DNS such as used on Thread the service domain SHALL be as configured automatically by the Thread Border Router.

4.3.1.1. Host Name Construction

For DNS'SD a target host name is required, in addition to the instance name. The target host name SHALL be constructed using one of the available link-layer addresses, such as a 48-bit device MAC address (for Ethernet and Wi'Fi) or a 64-bit MAC Extended Address (for Thread) expressed as a fixed-length twelve-character (or sixteen-character) hexadecimal string, encoded as ASCII (UTF-8) text using capital letters, e.g., B75AFB458ECD.<domain>. In the event that a device performs MAC address randomization for privacy, then the target host name SHALL use the privacy-preserving randomized version and the hostname SHALL be updated in the record every time the underlying link-layer address rotates. Note that it is legal to reuse the same hostname on more than one inter!face, even if the underlying link-layer address does not match the hostname on that interface, since the goal of using a link-layer address is to ensure local uniqueness of the generated hostname. If future link layers are supported by Matter that do not use 48-bit MAC addresses or 64-bit MAC Extended Address identifiers, then a similar rule will be defined for those technologies.

4.3.1.2. Extended Discovery

A Matter Commissionee that advertises Commissionable Node Discovery service records is not nec!essarily in a state that will allow Commissioning (this state is referred to below as "in Commission!ing Mode"). While Section 5.4.2.3, ÒAnnouncement DurationÓ is limited for some forms of device advertisement, a Matter device MAY advertise Matter Commissionable Node Discovery service records for longer periods, possibly permanently. Advertising Commissionable Node Discovery

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

when not in Commissioning Mode is referred to here as Extended Discovery. Extended Discovery is allowed only for DNS-SD advertisements and not for the other forms of Device Discovery such as BLE Commissioning Discovery and Wi-Fi Temporary Access Point Commissioning Discovery .

To protect customer privacy on public networks, a Matter Commissionee SHALL provide a way for the customer to set a timeout on Extended Discovery, or otherwise disable Extended Discovery. The default behavior for Commissionable Node Discovery SHOULD default to limiting announcement as defined in Section 5.4.2.3, ÒAnnouncement DurationÓ unless the Manufacturer wishes to enable longer periods for specific use cases.

4.3.1.3. Commissioning Subtypes

The following subtypes for Matter Commissionable Node Discovery are defined:

1. _L<dddd>, where <dddd> provides the full 12-bit discriminator , encoded as a variable-length deci! mal number in ASCII text, omitting any leading zeroes.
2. _S<dd>, where <dd> provides the upper 4 bits of the discriminator , encoded as a variable-length decimal number in ASCII text, omitting any leading zeroes.
3. _V<ddddd>, where <ddddd> provides the 16-bit Vendor ID , encoded as a variable-length decimal number in ASCII text, omitting any leading zeroes.
4. _T<ddd>, where <ddd> provides the device type identifier for the device, encoded as a variable-length decimal number in ASCII (UTF-8) text, omitting any leading zeroes. In case the device combines multiple device types, the manufacturer SHOULD choose the device type identifier of the primary function of the device for which the device wishes to be discoverable.
5. _CM, which represents "currently in Commissioning Mode" (due to any method, for example, a factory new device that has just been put into commissioning mode by the user, or an already-commissioned device which has just received the Open Commissioning Window command).

The long discriminator subtype (e.g., _L840) enables filtering of results to find only Commissionees that match the full discriminator code, as provided in the onboarding payload .

The short discriminator subtype (e.g., _S3) enables filtering of results to find only Commissionees that match the upper 4 bits of the discriminator code, as provided in the manual pairing code .

The optional Vendor ID subtype (e.g., _V123) enables a vendor-specific app to achieve filtering of results to find only Nodes that match that Vendor ID.

The Commissioning Mode subtype (e.g., _CM) enables filtering of results to find only Nodes that are currently in Commissioning Mode. Note that the subtype is _CM regardless of whether the TXT record for commissioning mode is set to 1 (CM=1) or 2 (CM=2). A Commissionee that is not in commis! sioning mode (CM=0) SHALL NOT publish this subtype.

The optional device type subtype (e.g., _T10) enables filtering of results to find only Nodes that match the device type, generally used for the User-Initiated Beacon Detection, Not Yet Commissioned Device and the User-Initiated Beacon Detection, Already Commissioned Device use cases.

In the event that a vendor-specific app wishes to show the user only some of that vendorÕs Commis! sionees awaiting commissioning but not all of them, any desired filtering logic (based upon arbi!

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

trary criteria, not only Product ID) MAY be implemented within that vendorÕs proprietary commis! sioning app.

4.3.1.4. TXT Records

After discovery, IPv6 addresses are returned in the AAAA records and key/value pairs are returned in the DNS'SD TXT record.

Nodes SHALL publish AAAA records for all available IPv6 addresses upon which they are willing to accept Matter commissioning messages.

TXT records available for Commissionable Node Discovery include the common TXT record key/value pairs defined in Section 4.3.4, ÒCommon TXT Key/Value PairsÓ.

Commissioners SHALL silently ignore TXT record keys that they do not recognize. This is to facili! tate future evolution of this specification without breaking backwards compatibility with existing Commissioners that do not implement the new functionality.

The following subsections describe key/value pairs that are defined specifically for Commissionable Node discovery.

4.3.1.5. TXT key for discriminator (D)

The key D SHALL provide the full 12-bit discriminator for the Commissionable Node and SHALL be present in the DNS-SD TXT record.

The discriminator value SHALL be encoded as a variable-length decimal number in ASCII text, with up to four digits, omitting any leading zeroes.

Any key D with a value mismatching the aforementioned format SHALL be silently ignored.

As an example, value D=840 would indicate that this Commissionable Node has decimal long dis! criminator 840. When needed, the upper 4 bits of the discriminator provided by the manual pairing code can be algorithmically derived from the full discriminator.

4.3.1.6. TXT key for Vendor ID and Product ID (VP)

The optional key VP, if present, MAY provide Vendor ID and Product ID information of the device.

A vendor MAY choose not to include it at all, for privacy reasons.

If the VP key is present then it MAY take two forms:

1. VP=123 gives Vendor ID
2. VP=123+456 gives Vendor ID + Product ID

The Vendor ID and Product ID SHALL both be expressed as variable-length decimal numbers, encoded as ASCII text, omitting any leading zeroes, and of maximum length of 5 characters each to fit their 16-bit numerical range.

If the Product ID is present, it SHALL be separated from the Vendor ID using a Ô+Õ character.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

If the VP key is present without a Product ID, the value SHALL contain only the Vendor ID alone, with no Ô+Õ character.

If the VP key is present, the value SHALL contain at least the Vendor ID.

If the VP key is present, it SHALL NOT have a missing or empty value.

4.3.1.7. TXT key for commissioning mode (CM)

The key CM (Commissioning Mode) SHALL indicate whether or not the publisher of the record is currently in Commissioning Mode and available for immediate commissioning. When in commissioning mode, the value associated with the CM key indicates the source of the passcode.

Four situations are legal:

1. The absence of key CM SHALL imply a value of 0 (CM=0).
2. The key/value pair CM=0 SHALL indicate that the publisher is not currently in Commissioning Mode.
3. The key/value pair CM=1 SHALL indicate that the publisher is currently in Commissioning Mode and requires use of a passcode for commissioning provided by the Commissionee (e.g., printed on a label or displayed on screen), such as when the device is in a factory-new state or when the Open Basic Commissioning Window command has been used to enter commissioning mode.
4. The key/value pair CM=2 SHALL indicate that the publisher is currently in Commissioning Mode and requires use of a dynamically generated passcode for commissioning corresponding to the verifier that was passed to the device using the Open Commissioning Window command.

A key value of 2 MAY be used to disambiguate collisions of discriminators between uncommissioned Nodes and commissioned Nodes announcing after a commissioning window was opened. A key value of 2 serves as a hint to Commissioners to possibly expect multiple Nodes with the same discriminator (see Commissioning Discriminator), and to instruct the user to enter the Onboarding Payload presented by another Administrator rather than a code provided by the Commissionee.

Since Extended Discovery can be disabled by the customer, a key value of 0 may not ever be returned by a publisher. When Extended Discovery is disabled and the publisher is not in commissioning mode, then the publisher will not respond to Commissionable Node Discovery.

4.3.1.8. TXT key for device type (DT)

The optional key DT MAY provide the publisherÕs primary device type (see Section 11.22.5.3, ÒDevice TypeIDÓ). In case the device combines multiple device types, the manufacturer SHOULD choose the device type identifier of the primary function of the device for which the device wishes to be discoverable. If present, it SHALL be encoded as a variable-length decimal number in ASCII text, omitting any leading zeroes.

For example, the DT=10 key/value pair would indicate that the primary device type is 10 (0x000a), which is the device type identifier for a Door Lock.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

4.3.1.9. TXT key for device name (DN)

The optional key DNMAY provide a device advertisement name. If present, it SHALL be encoded as a valid UTF-8 string with a maximum length of 32 bytes (matching the maximum length of the Node! Label string in the Basic Information Cluster).

When provided, the source of this value SHALL be editable by the user with use clearly designated as being for on-network advertising and MAY be the value stored in the NodeLabel attribute of the Basic Information Cluster) of the Node.

To protect customer privacy on public networks, if a Commissionee supports this key/value pair, then the Commissionee SHALL provide a way for the customer to disable its inclusion.

A Commissionee SHOULD NOT include this field unless doing so for specific use cases which call for it.

For example, the DN=Living Roomkey/value pair indicates that the advertisement name specified by the user is 'Living Room'.

4.3.1.10. TXT key for rotating device identifier (RI)

The optional key RI MAY provide a Rotating Device Identifier .

If present, the value associated with the RI key SHALL contain the octets of the Rotating Device Identifier octet string encoded as the concatenation of each octetÕs value as a 2-digit uppercase hexadecimal number.

The resulting ASCII string SHALL NOT be longer than 100 characters, which implies a Rotating Device Identifier of at most 50 octets.

4.3.1.11. TXT key for pairing hint (PH)

The optional key PHMAY provide a pairing hint .

If present, it SHALL be encoded as a variable-length decimal number in ASCII text, omitting any leading zeroes.

The pairing hint represents a base-10 numeric value for a bitmap of methods supported by the Commissionee in its current state for putting it in Commissioning Mode.

For example, the PH=5key/value pair represents a hint value with bits 0 and 2 are set.

This value MAY change during the lifecycle of the device.

For example, a device may have a value with bit 0 (Power Cycle) set and bit 2 (Administrator app) unset when in a factory reset state, and then have a value with bit 0 unset and bit 2 set after it has been Commissioned.

The bitmap of methods is defined in Table 6, ÒPairing Hint ValuesÓ .

If the Commissionee has enabled Extended Discovery , then it SHALL include the key/value pair for PHin the DNS'SD TXT record when not in Commissioning Mode (CM=0

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

This key/value pair MAY be returned when in Commissioning Mode (CM=1).

If the Commissioner does not recognize this value, for example, if the value indicates bit indices defined in a newer version of this specification than the version which the Commissioner implements, then the Commissioner MAY utilize the bits that it does understand and MAY utilize additional data sets available for assisting the user. For example, when a Vendor ID and Product ID are available to the Commissioner, the Section 11.22, ÒDistributed Compliance LedgerÓ may also provide a URL for the Device User Guide which can contain additional information to help in Commissioning this Commissionee.

Some of the pairing hints MAY require additional information to be encoded for proper expression of their meaning. This is accomplished with the PI TXT key, described in a following section. Dependency on usage of the PI key is expressed by the PI Dependency column in the table below.

The following fields in the bitmap are currently defined for values of the PH key:

Table 6. Pairing Hint Values

Bit index	Name	PI Dependency	Description
0	Power Cycle	False	The Device will automatically enter Commissioning Mode upon power cycle (unplug/replug, remove/re-insert batteries). This bit SHALL be set to 1 for devices using Standard Commissioning Flow , and set to 0 otherwise.
1	Device Manufacturer URL	False	This SHALL be set to 1 for devices requiring Custom Commissioning Flow before they can be available for Commissioning by any Commissioner. For such a flow, the user SHOULD be sent to the URL specified in the CommissioningCustomFlowUrl of the DeviceModel schema entry indexed by the Vendor ID and Product ID (e.g., as found in the announcement) in the Distributed Compliance Ledger .

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Bit index	Name	PI Dependency	Description
2	Administrator	False	The Device has been commissioned. Any Administrator that commissioned the device provides a user interface that may be used to put the device into Commissioning Mode.
3	Settings menu on the Node	False	The settings menu on the Device provides instructions to put it into Commissioning Mode.
4	Custom Instruction	True	The PI key/value pair describes a custom way to put the Device into Commissioning Mode. This Custom Instruc! tion option is NOT rec! ommended for use by a Device that does not have knowledge of the userÕs language prefer! ence.
5	Device Manual	False	The Device Manual pro! vides special instruc! tions to put the Device into Commissioning Mode (see Section 11.22.5.8, ÒUserManu! alUrlÓ). This is a catch-all option to capture user interactions that are not codified by other options in this ta! ble.
6	Press Reset Button	False	The Device will enter Commissioning Mode when reset button is pressed.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Bit index	Name	PI Dependency	Description
7	Press Reset Button with application of power	False	The Device will enter Commissioning Mode when reset button is pressed when applying power to it.
8	Press Reset Button for N seconds	True	The Device will enter Commissioning Mode when reset button is pressed for N seconds. The exact value of N SHALL be made avail! able via PI key.
9	Press Reset Button until light blinks	True	The Device will enter Commissioning Mode when reset button is pressed until associated light blinks. Informa! tion on color of light MAY be made available via PI key (see Note 1).
10	Press Reset Button for N seconds with applica! tion of power	True	The Device will enter Commissioning Mode when reset button is pressed for N seconds when applying power to it. The exact value of N SHALL be made available via PI key.
11	Press Reset Button until light blinks with appli! cation of power	True	The Device will enter Commissioning Mode when reset button is pressed until associated light blinks when applying power to the Device. Information on color of light MAY be made available via PI key (see Note 1).

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Bit index	Name	PI Dependency	Description
12	Press Reset Button N times	True	The Device will enter Commissioning Mode when reset button is pressed N times with maximum 1 second between each press. The exact value of N SHALL be made avail! able via PI key.
13	Press Setup Button	False	The Device will enter Commissioning Mode when setup button is pressed.
14	Press Setup Button with application of power	False	The Device will enter Commissioning Mode when setup button is pressed when applying power to it.
15	Press Setup Button for N seconds	True	The Device will enter Commissioning Mode when setup button is pressed for N seconds. The exact value of N SHALL be made avail! able via PI key.
16	Press Setup Button until light blinks	True	The Device will enter Commissioning Mode when setup button is pressed until associated light blinks. Informa! tion on color of light MAY be made available via PI key (see Note 1).
17	Press Setup Button for N seconds with applica! tion of power	True	The Device will enter Commissioning Mode when setup button is pressed for N seconds when applying power to it. The exact value of N SHALL be made available via PI key.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Bit index	Name	PI Dependency	Description
18	Press Setup Button until light blinks with application of power	True	The Device will enter Commissioning Mode when setup button is pressed until associated light blinks when applying power to the Device. Information on color of light MAY be made available via PI key (see Note 1).
19	Press Setup Button N times	True	The Device will enter Commissioning Mode when setup button is pressed N times with maximum 1 second between each press. The exact value of N SHALL be made avail! able via PI key.

Note 1: When the PHkey indicates a light to blink (one or more of bits 9, 11, 16 or 18 is set), informa! tion on color of light MAY be made available via PI key. When using such color indication in PI key, only basic primary and secondary colors that could unambiguously be decoded by a commissioner and understood by an end-user, but without worry of localization, SHOULD be used, e.g. white, red, green, blue, orange, yellow, purple.

Note 2: Any undefined values are reserved for future use.

Note 3: A Commissionee can indicate multiple ways of being put into Commissioning Mode by set! ting multiple bits in the bitmap at the same time. However, only one method can be specified which has a dependency on the PI key (PI Dependency=True) at a time.

For example:

- ¥ A PHvalue of 33 (bits 0 and 5 are set) indicates that the user can cause the Commissionee to enter Commissioning Mode by either power cycling it or by following special instructions pro! vided in the Device Manual.
- ¥ A PHvalue of 9 (bits 0 and 3 are set) indicates that the user can cause the Commissionee to enter Commissioning Mode by either power cycling it or going to the settings menu and following instructions there.
- ¥ A PHvalue of 1 (bit 0 is set) indicates that the user can cause the Commissionee to enter Commis! sioning Mode only by power cycling it.
- ¥ A PHvalue of 16 (bit 4 is set) indicates that the user can cause the Commissionee to enter Com! missioning Mode following a custom procedure described by the value of the PI key.
- ¥ A PHvalue of 256 (bits 8 is set) indicates that the user can cause the Commissionee to enter Com!

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

missioning Mode by pressing the reset button for a duration of time in seconds specified via by the value of the PI key.

When the PH key is provided, at least one bit in the above bitmap SHALL be set. That is, a PH value of 0 is undefined and illegal.

When the PH key is provided, the Commissioner SHOULD take its value into account when provid! ing guidance to the user regarding steps required to put the Commissionee into Commissioning Mode.

4.3.1.12. TXT key for pairing instructions (PI)

The optional key PI MAY give the pairing instruction .

If present, the value SHALL be encoded as a valid UTF-8 string with a maximum length of 128 bytes.

The meaning of this key is dependent upon the PH key value, see Table 6, ÒPairing Hint ValuesÓ .

For example, given PH=256 bit 8 is set which indicates "Press Reset Button for N seconds". Therefore, a value of PI=10 would indicate that N is 10 in that context.

When bit 4 of the value expressed by the PH key is set, indicating presence of a custom string, the Commissionee SHALL be responsible for localization (translation to userÕs preferred language) as required using the DeviceÕs currently configured locale. The Custom Instruction option is NOT rec! ommended for use by a Commissionee that does not have knowledge of the userÕs language prefer! ence.

It is RECOMMENDED to keep the length of PI field small and adhere to the guidance given in section 6.2 of [RFCÊ6763].

This key/value pair SHALL only be returned in the DNS'SD TXT record if the PH bitmap value has a bit set which has PI Dependency = True, see Table 6, ÒPairing Hint ValuesÓ . The PH key SHALL NOT not have more than one bit set which has a dependency on the PI key (PI Dependency = True) to avoid ambiguity in PI key usage.

4.3.1.13. Examples

The examples below simulate a Node in commissioning mode advertising its availability for com! missioning.

Examples are shown using both the dns-sd command-line test tool and the avahi command-line test tool. The dns-sd command-line test tool is included in all versions of macOS. It is installed as a DOS command with Bonjour for Windows, and is available on Linux by installing the mDNSResponder package [https://github.com/balaji-reddy/mDNSResponder]. The Avahi package of command line tools is available from the Avahi project [https://github.com/lathiat/avahi)] for most Linux distributions.

These examples are given for illustrative purposes only. Real Matter Commissionees and Commis! sioners would not use a command-line test tool for advertising and discovery. Real Matter Commis! sionees and Commissioners would use the appropriate DNS'SD APIs in their respective chosen pro! gramming languages.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Consider a device on Wi-Fi using the 48-bit device MAC address of B75AFB458ECD as its host name and a value of DD200C20D25AE5F7 as its commissionable service instance name. DNS-SD records for it can be set up as follows:

```
dns-sd -R DD200C20D25AE5F7 _matterc._udp,_S3,_L840,_CM . 11111 D=840 CM=2
```

or

```
avahi-publish-service --subtype=_S3._sub._matterc._udp
--subtype=_L840._sub._matterc._udp DD200C20D25AE5F7 --subtype=_CM._sub._matterc._udp
_matterc._udp 11111 D=840 CM=2
```

- ¥ Short discriminator is filterable through _S3 subtype and algorithmically through D=840 TXT key.
- ¥ Long discriminator is filterable through _L840 subtype and directly through D=840 TXT key.
- ¥ The Commissionee is currently in Commissioning Mode after an Administrator having opened a commissioning window (see Section 4.3.1.7, ÒTXT key for commissioning mode (CMÓ), as shown by CM=2 TXT key and availability by browsing the _CM subtype.

  %Had the Commissionee been discoverable for initial commissioning rather than subsequent additional commissioning, a CM=1 TXT key would have been published instead.

Avahi only sends a single AAAA record. To force the link-local address to be used, use avahi-pub! lish-address. For example:

```
avahi-publish-address B75AFB458ECD.local fe80::f515:576f:9783:3f30
```

The DNS'SD service registration commands shown above results in the creation of the following Multicast DNS records:

```
_matterc._udp.local.                PTR     DD200C20D25AE5F7._matterc._udp.local.
_S3._sub._matterc._udp.local.       PTR     DD200C20D25AE5F7._matterc._udp.local.
_L840._sub._matterc._udp.local.     PTR     DD200C20D25AE5F7._matterc._udp.local.
_CM._sub._matterc._udp.local.       PTR     DD200C20D25AE5F7._matterc._udp.local.
DD200C20D25AE5F7._matterc._udp.local.  SRV     0 0 11111 B75AFB458ECD.local.
DD200C20D25AE5F7._matterc._udp.local.  TXT     "D=840" "CM=2"
B75AFB458ECD.local.                 AAAA    fe80::f515:576f:9783:3f30
```

Consider a device on Wi-Fi using the 48-bit device MAC address of B75AFB458ECD as its host name. DNS-SD records for it can be set up as follows, when it is in Commissionable Node Discovery.

```
dns-sd -R DD200C20D25AE5F7 _matterc._udp,_S3,_L840,_V123,_CM,_T81 . 11111 D=840
VP=123+456 CM=2 DT=81 DN="Kitchen Plug" PH=256 PI=5
```

or

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

```
avahi-publish-service --subtype=_S3._sub._matterc._udp
--subtype=_L840._sub._matterc._udp --subtype=_V123._sub._matterc._udp
--subtype=_CM._sub._matterc._udp --subtype=_T81._sub._matterc._udp DD200C20D25AE5F7
_matterc._udp 11111 D=840 VP=123+456 CM=2 DT=81 DN="Kitchen Plug" PH=256 PI=5
```

- Short discriminator is 3, long discriminator is 840.
- Vendor ID is 123, Product ID is 456.
- Commissioning Mode is 2, indicating the Commissionee is currently in Commissioning Mode due to the Open Commissioning Window command.
- Device type is 81 which is a Smart Plug (Device Type Identifier 0x0051).
- Device name is Kitchen Plug .
- Pairing hint is 256 which indicates that the CommissioneeÕs reset button must be held down for 5 seconds to enter Commissioning Mode where the value 5 is obtained by reading the value of the PI key.
- Pairing instruction is 5.

Avahi only sends a single AAAA record. To force the link-local address to be used, use avahi-pub! lish-address. For example:

```
avahi-publish-address B75AFB458ECD.local fe80::f515:576f:9783:3f30
```

The DNS'SD service registration commands shown above results in the creation of the following Multicast DNS records:

```
_matterc._udp.local.                PTR   DD200C20D25AE5F7._matterc._udp.local.
_S3._sub._matterc._udp.local.       PTR   DD200C20D25AE5F7._matterc._udp.local.
_L840._sub._matterc._udp.local.     PTR   DD200C20D25AE5F7._matterc._udp.local.
_V123._sub._matterc._udp.local.     PTR   DD200C20D25AE5F7._matterc._udp.local.
_CM._sub._matterc._udp.local.       PTR   DD200C20D25AE5F7._matterc._udp.local.
_T81._sub._matterc._udp.local.      PTR   DD200C20D25AE5F7._matterc._udp.local.
DD200C20D25AE5F7._matterc._udp.local.  TXT   "D=840" "VP=123+456" "CM=1" "DT=81"
"DN=Kitchen Plug" "PH=256" "PI=5"
DD200C20D25AE5F7._matterc._udp.local.  SRV   0 0 11111 B75AFB458ECD.local.
B75AFB458ECD.local.                 AAAA  fe80::f515:576f:9783:3f30
```

The port number 11111 is given here purely as an example. One of the benefits of using DNS'SD is that services are not constrained to use a single predetermined well-known port. The port, along with the IPv6 address, is discovered by Commissioners at run time.

A Commissioner can discover all available Commissionees awaiting commissioning:

```
dns-sd -B _matterc._udp
```


Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

or

```
avahi-browse _matterc._udp -r
```

A Commissioner can discover Commissionees awaiting commissioning with short discriminator 3:

```
dns-sd -B _matterc._udp,_S3
```

or

```
avahi-browse _S3._sub._matterc._udp -r
```

A Commissioner can discover Commissionees awaiting commissioning with long discriminator 840:

```
dns-sd -B _matterc._udp,_L840
```

or

```
avahi-browse _L840._sub._matterc._udp -r
```

A Commissioner can discover Commissionees awaiting commissioning with Vendor ID 123:

```
dns-sd -B _matterc._udp,_V123
```

or

```
avahi-browse _V123._sub._matterc._udp -r
```

A Commissioner can discover all Commissionees in commissioning mode:

```
dns-sd -B _matterc._udp,_CM
```

or

```
avahi-browse _CM._sub._matterc._udp -r
```

A commissioner can discover Matter Nodes with Device Type 81:

```
dns-sd -B _matterc._udp,_T81
```

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

or

```
avahi-browse _T81._sub._matterc._udp -r
```

A Commissioner can discover Nodes that are currently in Commissioning Mode as a result of a com! missioning window opened by a current Administrator as a result of invoking either the Open Com! missioning Window command or the Open Basic Commissioning Window command, using the presence of the _CM subtype as a browsing filter:

```
dns-sd -B _matterc._udp,_CM
```

or

```
avahi-browse _CM._sub._matterc._udp -r
```

4.3.1.14. Efficiency Considerations

Discovering and using an offered service on the network typically involves several steps:

1. Enumeration of instances available on the network ("browsing")
2. Lookup of a selected instanceÕs port number, host name, and other additional information, com! municated in DNS'SD using SRV and TXT records ("resolving")
3. Lookup of the IPv6 address(es) associated with the desired target host.
4. Use of IPv6 Neighbor Discovery and/or IPv6 routing to translate from destination IPv6 address to the next-hop link-layer address for that communication.
5. Establishing a cryptographically secure communication channel between the two endpoints, and then engaging in useful communication.

Although the first three steps are exposed in some APIs as separate steps, at a protocol level they usually require only a single network round-trip. When a PTR query is issued to discover service instances, the usual DNS Additional Record mechanism, where packet space permits, automatically places the related SRV, TXT, and address records into the Additional Record section of the reply. These additional records are stored by the client, to enable subsequent steps in the sequence to be performed without additional redundant network operations to learn the same information a sec! ond time.

DNS'SD over Multicast DNS works by receiving replies from other Nodes attached to the same local link, Nodes that may have been previously completely unknown to the requester. Because of this, Multicast DNS, like IPv6 Neighbor Discovery, does not have any easy way to distinguish genuine replies from malicious or fraudulent replies. Consequently, application-layer end-to-end security is essential. Should a malicious device on the same local link give deliberately malicious or fraudulent replies, the misbehavior will be detected when the device is unable to establish a cryptographically secure application-layer communication channel. This reduces the threat to a Denial-of-Service attack, which can be remedied by physically removing the offending device.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

4.3.2. Operational Discovery

For Matter Nodes that have already been commissioned onto a Matter Fabric, run-time dynamic discovery of operational Matter Nodes is used, rather than assuming a fixed unchanging IPv6 address and port for the lifetime of the product. This is done to allow for greater flexibility, so that the underlying IPv6 network can grow and evolve over time as needed without breaking Matter functionality. This is the same reason that other networked consumer electronics products do not assume a single fixed unchanging IP address for the lifetime of the product [RFCÊ5505].

4.3.2.1. Operational Instance Name

For Matter operational discovery the DNS'SD instance name is constructed from a 64-bit com! pressed Fabric identifier , and a 64-bit Node identifier, as assigned by the commissioner, each expressed as a fixed-length sixteen-character hexadecimal string, encoded as ASCII (UTF-8) text using capital letters, separated by a hyphen. For example, a Matter Node with Matter compressed fabric identifier 2906-C908-D115-D362 and Matter Node identifier 8FC7-7724-01CD-0696 has Matter operational discovery DNS'SD instance name 2906C908D115D362-8FC7724401CD0696

The Matter operational discovery DNS'SD instance name needs to be unique within the namespace of the local network (the .local link-local namespace of the Ethernet and Wi'Fi links [RFCÊ6762], or the unicast domain selected by the Thread Border Router for devices on the Thread mesh). This uniqueness is assumed to be guaranteed by appropriate selection of a unique Matter fabric identi! fier and unique Node identifier within that Matter fabric.

4.3.2.2. Compressed Fabric Identifier

In order to reduce the very large size of a full Fabric Reference which would need to be used as the scoping construct in the instance name , a 64-bit compressed version of the full Fabric Reference SHALL be used. The computation of the Compressed Fabric Identifier SHALL be as follows:

```
byte CompressedFabricInfo[16] = /* "CompressedFabric" */
Ê    {0x43, 0x6f, 0x6d, 0x70, 0x72, 0x65, 0x73, 0x73,
Ê     0x65, 0x64, 0x46, 0x61, 0x62, 0x72, 0x69, 0x63}

CompressedFabricIdentifier =
Ê    Crypto_KDF(
Ê      inputKey := TargetOperationalRootPublicKey,
Ê      salt:= TargetOperationalFabricID,
Ê      info := CompressedFabricInfo,
Ê      len := 64)
```

Where:

¥ TargetOperationalRootPublicKey is the raw uncompressed elliptical curve public key of the root certificate for the advertised NodeÕs Operational Certificate chain, without any format marker prefix byte (i.e. after removing the first byte of the ec-pub-key field in the Operational Certifi! cateÕs root).

¥ TargetOperationalFabricID is the octet string for the Fabric ID as it appears in the advertised

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

NodeÕs Operational Certificate 's subject field, under the 1.3.6.1.4.1.37244.1.5 RDN, that is, a 64-bit unsigned integer scalar in big-endian byte order.

For example, if the root public key for a given Operational Certificate chain containing the identity to be advertised were the following:

```
pub:
Ê 04:4a:9f:42:b1:ca:48:40:d3:72:92:bb:c7:f6:a7:e1:
Ê 1e:22:20:0c:97:6f:c9:00:db:c9:8a:7a:38:3a:64:1c:
Ê b8:25:4a:2e:56:d4:e2:95:a8:47:94:3b:4e:38:97:c4:
Ê a7:73:e9:30:27:7b:4d:9f:be:de:8a:05:26:86:bf:ac:
Ê fa
```

Then the value for TargetOperationalRootPublicKey to use in the derivation of the compressed Fab! ric Identifier would be without the leading 04:

```
Ê 4a:9f:42:b1:ca:48:40:d3:72:92:bb:c7:f6:a7:e1:1e:
Ê 22:20:0c:97:6f:c9:00:db:c9:8a:7a:38:3a:64:1c:b8:
Ê 25:4a:2e:56:d4:e2:95:a8:47:94:3b:4e:38:97:c4:a7:
Ê 73:e9:30:27:7b:4d:9f:be:de:8a:05:26:86:bf:ac:fa
```

If using the above TargetOperationalRootPublicKey and a TargetOperationalFabricID value of 0x2906_C908_D115_D362 (octet string 29:06:c9:08:d1:15:d3:62 in big-endian), then the Compressed! FabricIdentifier to use in advertising would be 87E1B004E235A130 (octet string 87:e1:b0:04:e2:35:a1:30).

4.3.2.3. Operational Service Type

The DNS'SD service type [RFCÊ6335] for Matter Operational Discovery is _matter._tcp . Note that the string _tcp is boilerplate text inherited from the original DNS SRV specification [RFCÊ2782], and doesnÕt necessarily mean that the advertised application-layer protocol runs only over TCP. It is merely mnemonic text which is there to help human readers, and in no way affects software adver! tising or using the application-layer protocol identified by that unique IANA-recorded service type string.

The following subtype is defined:

1. Compressed Fabric Identifier _I<hhhh>, where <hhhh> is the Compressed Fabric Identifier encoded as exactly 16 uppercase hexadecimal characters, for example _I87E1B004E235A130 for the Compressed Fabric Identifier example of the previous section. This subtype enables filtering of devices per Fabric if service enumeration (browsing) is attempted, to reduce the set of results to Nodes of interest with operational membership in a given Fabric..

4.3.2.4. Operational Service Domain and Host Name

For link-local Multicast DNS the service domain SHALL be local . For Unicast DNS such as used on Thread the service domain SHALL be as configured automatically by the Thread Border Router.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

For DNS'SD a target host name is required, in addition to the instance name. The target host name SHALL be constructed using one of the available link-layer addresses, such as a 48-bit device MAC address (for Ethernet and Wi'Fi) or a 64-bit MAC Extended Address (for Thread) expressed as a fixed-length twelve-character (or sixteen-character) hexadecimal string, encoded as ASCII (UTF-8) text using capital letters, e.g., B75AFB458ECD.<domain>. In the event that a device performs MAC address randomization for privacy, then the target host name SHALL use the privacy-preserving randomized version and the hostname SHALL be updated in the record every time the underlying link-layer address rotates. Note that it is legal to reuse the same hostname on more than one inter! face, even if the underlying link-layer address does not match the hostname on that interface, since the goal of using a link-layer address is to ensure local uniqueness of the generated hostname. If future link layers are supported by Matter that do not use 48-bit MAC addresses or 64-bit MAC Extended Address identifiers, then a similar rule will be defined for those technologies.

4.3.2.5. Operational Discovery Service Records

After discovery, IPv6 addresses are returned in the AAAA records and key/value pairs are returned in the DNS'SD TXT record. The TXT record MAY be omitted if no keys are defined.

Nodes SHALL publish AAAA records for all available IPv6 addresses upon which they are willing to accept operational messages.

Only the common TXT record key/value pairs defined in Section 4.3.4, ÒCommon TXT Key/Value PairsÓ are defined for use in Operational Discovery.

Nodes SHALL silently ignore TXT record keys that they do not recognize.

4.3.2.6. Performance Recommendations

To improve overall performance of operational discovery, especially in large installations, the fol! lowing recommendations SHOULD be taken in account:

1. Nodes SHOULD cache the last-known IPv6 address and port for each peer Node with which they interact from their SRV record resolved using DNS-SD, to save the cost of a run-time network lookup operation when not needed. When the IPv6 address and port for a peer Node is not known, or an attempt to communicate with a peer Node at its last-known IPv6 address and port does not appear to be succeeding within the expected network round-trip time (i.e., the retrans! mission timeout value for the first message packet) a Node SHOULD then perform a run-time discovery in parallel, to determine whether the desired Node has acquired a new IPv6 address and/or port [RFCÊ8305].
2. Nodes SHOULD respond to nonspecific service enumeration queries for the generic Matter Operational Discovery service type (_matter._tcp), but these queries SHOULD NOT be used in routine operation, and instead it is RECOMMENDED that they only be used for diagnostics pur! poses or to determine new membership within a fabric. When used, it is RECOMMENDED that service enumeration employ the _I<HHHH> Fabric-specific subtype to only enumerate the desired Nodes on the Fabric of interest in the local network. Moreover, Known Answer Suppression [RFCÊ6762] SHOULD be employed in such cases to further minimize the number of unnecessary responses to such a query.
3. When resolving the operational service record of another Node, a Node SHOULD use an SRV

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

query for the desired operational service instance rather than doing general enumeration of all nodes (e.g. PTR query) followed by filtering for the desired service instance. This recommendation reduces the amount of multicast traffic generated on-link when Multicast DNS is used, and reduces latency to successful service resolution.

4. Since proxied DNS-SD service discovery MAY be in use within a given network, and service record caching is expected of DNS-SD clients, Nodes SHOULD NOT use DNS-SD as an operational liveness determination method. This is because there may be stale records not yet expired after a Node becomes unreachable which may still be available.

4.3.2.7. Operational Discovery DNS-SD Examples

The example below simulates a commissioned Matter Node advertising its availability for control via the Matter protocol.

Examples are shown using both the dns-sd command-line test tool and the avahi command-line test tool. The dns-sd command-line test tool is included in all versions of macOS. It is installed as a DOS command with Bonjour for Windows, and is available on Linux by installing the mDNSResponder package [https://github.com/balaji-reddy/mDNSResponder]. The avahi command line-test tool is available from the Avahi project [https://github.com/lathiat/avahi)] for most Linux distributions.

This example is given for illustrative purposes only. Real Matter Nodes and controllers would not use a command-line test tool for advertising and discovery. Real Matter Nodes and controllers would use the appropriate DNS'SD APIs in their respective chosen programming languages.

Consider a device on Wi-Fi using the 48-bit device MAC address of B75AFB458ECD as its host name. DNS-SD records for can be set up as follows:

```
dns-sd -R 87E1B004E235A130-8FC7772401CD0696 _matter._tcp . 22222
```

or

```
avahi-publish-service 87E1B004E235A130-8FC7772401CD0696 _matter._tcp 22222
```

The port number 22222 is given here purely as an example. One of the benefits of using DNS'SD is that services are not constrained to use a single predetermined well-known port. This means that multiple instances of the Matter Node control service can run on the same device at the same time, listening on different ports [RFCÊ6760 The port, along with the IPv6 address, is discovered by the Matter controller at run time.

Avahi only sends a single AAAA record. To force the link-local address to be used, use avahi-publish-address. For example:

```
avahi-publish-address B75AFB458ECD.local fe80::f515:576f:9783:3f30
```

A Matter controller can discover the current IPv6 address and port for a known commissioned Matter Node:

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

```
dns-sd -L 87E1B004E235A130-8FC7772401CD0696 _matter._tcp
87E1B004E235A130-8FC7772401CD0696._matter._tcp.local. can be reached at
B75AFB458ECD.local.:22222

dns-sd -Gv6 B75AFB458ECD.local
fe80::f515:576f:9783:3f30
```

or

```
avahi-browse _matter._tcp -r

hostname = [B75AFB458ECD.local]
address = [fe80::f515:576f:9783:3f30]
port = [22222]
```

4.3.3. Commissioner Discovery

A Commissionee MAY initiate the commissioning process by discovering Commissioners on the net! work (see Initiating Commissioning from an Existing Device). This MAY be done using Commis! sioner Discovery described in this section.

With Commissioner Discovery, a Commissionee, upon user interaction, MAY discover Commission! ers on the network and obtain a list of information for each which may include Vendor ID, Product ID and friendly name. A Commissionee with a user interface, such as a Television, Thermostat or Video Player device, MAY display the list of discovered commissioners to the user for selection. Once selected, the Commissionee MAY use the User Directed Commissioning protocol with the Com! missioner to indicate that the user has selected it for commissioning of the Commissionee. The Commissioner Discovery service records thus enable a form of "door bell" protocol to allow a Com! missionee to request Commissioning.

The Commissioner Discovery feature is optional for both the Commissionee and the Commissioner. Any mandatory requirements described in this section SHALL apply only if the Node or the Com! missioner supports this feature. To protect customer privacy on public networks, a Matter Commis! sioner SHALL provide a way for the customer to set a timeout on Commissioner Discovery, or other! wise disable Commissioner Discovery.

For Commissioner Discovery, the DNS-SD instance name is generated the same way it is done for Commissionable Node Discovery and has the same requirements (uniqueness on local network, and collision detection and recovery) as those in Commissionable Node Discovery, but the require! ments for when a new instance name is selected from Commissionable Node Discovery do not apply to Commissioner Discovery. The instance name for Commissioner Discovery MAY be selected whenever the Commissioner deems necessary.

The DNS-SD service type [RFC6335] is _matterd._udp.

The port advertised by a _matterd._udp service record SHALL be different than any port associated with other advertised _matterc._udp, _matter._tcp or _matterd._udp services, in order to ensure that

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

the session-less messaging employed by the User Directed Commissioning protocol does not cause invalid message handling from fully operational Matter nodes at the same address. In other words, each _matterd._udp service instance needs to be independent from other services to ensure unam! biguous processing of the incoming User Directed Commissioning messages.

The following subtype is defined:

- _T<ddd>where <ddd>is the device type identifier (see Data Model Device Types), encoded as a variable-length decimal number in ASCII (UTF-8) text, without leading zeroes. This optional Device Type subtype enables filtering of results to find only Commissioners that match a device type, for example, to discover Commissioners of type Video Player (35 is decimal representation for Video Player device type identifier 0x0023). For such a Video Player filter, subtype _T35 would be used.

For link-local Multicast DNS the service domain SHALL be local . For Unicast DNS such as used on Thread the service domain SHALL be as configured automatically by the Thread Border Router.

The target host name is generated the same way it is done for Commissionable Node Discovery (see Host Name Construction).

After discovery, IPv6 addresses are returned in the AAAA records and key/value pairs are returned in the DNS'SD TXT record. The TXT record MAY be omitted if no keys are defined.

Nodes SHALL publish AAAA records for all their available IPv6 addresses.

In addition to the common TXT record key/value pairs defined in Section 4.3.4, ÒCommon TXT Key/Value PairsÓ, the following key/value pairs are defined specifically for Commissioner discovery:

- The optional key VPgives vendor and product information. This key is optional for a vendor to provide, and optional for a commissioner to use. This value takes the same format described for the VPkey in Commissionable Node Discovery (see Section 4.3.1.6, ÒTXT key for Vendor ID and Product ID (VP)Ó). This key/value pair MAY be returned in the DNS'SD TXT record.
- The optional key DT gives the device type identifier for the Commissioner (see Data Model Device Types). This value takes the same format described for the DTkey in Commissionable Node Discovery (see Commissioning Device Type). This key/value pair MAY be returned in the DNS'SD TXT record.
- The optional key DNgives the device name. This value takes the same format described for the DN key in Commissionable Node Discovery (see Commissioning Device Name). This key/value pair MAY be returned in the DNS'SD TXT record. To protect customer privacy on public networks, a Matter Commissioner SHALL provide a way for the customer to disable inclusion of this key.

Commissionees SHALL silently ignore TXT record keys that they do not recognize. This is to facili! tate future evolution of the Matter Commissioner Discovery protocol specification without breaking backwards compatibility with existing Commissionees that do not implement the new functionality.

4.3.3.1. Examples

The examples below simulate a Matter Commissioner advertising that it is present on the network.

Examples are shown using both the dns-sd command-line test tool and the avahi command-line test

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

tool. The dns-sd command-line test tool is included in all versions of macOS. It is installed as a DOS command with Bonjour for Windows, and is available on Linux by installing the mDNSResponder package [https://github.com/balaji-reddy/mDNSResponder]. The avahi command line-test tool is available from the Avahi project [https://github.com/lathiat/avahi)] for most Linux distributions.

These examples are given for illustrative purposes only.

Consider a device on Wi-Fi using the 48-bit device MAC address of B75AFB458ECD as its host name. DNS-SD records for can be set up as follows:

```
dns-sd -R DD200C20D25AE5F7 _matterd._udp,_V123,_T35 . 33333 VP=123+456 DT=35
DN="Living Room TV"
```

or

```
avahi-publish-service --subtype=_V123._sub._matterd._udp DD200C20D25AE5F7
_matterd._udp 33333 VP=123+456 DT=35 DN="Living Room TV"
```

This produces DNS-SD messages with the following characteristics:

¥ Vendor ID is 123, Product ID is 456.

¥ Device type is 35 which is a Video Player (Device Type Identifier 0x0023).

¥ Device name is Living Room TV.

Avahi only sends a single AAAA record. To force the link-local address to be used, use avahi-pub! lish-address. For example:

```
avahi-publish-address B75AFB458ECD.local fe80::f515:576f:9783:3f30
```

The DNS'SD service registration command shown above results in the creation of the following Multicast DNS records:

```
_matterd._udp.local.                PTR   DD200C20D25AE5F7._matterd._udp.local.
_V123._sub._matterd._udp.local.     PTR   DD200C20D25AE5F7._matterd._udp.local.
_T35._sub._matterd._udp.local.      PTR   DD200C20D25AE5F7._matterd._udp.local.
DD200C20D25AE5F7._matterd._udp.local.  TXT   "VP=123+456" "DT=35" "DN=Living Room TV"
DD200C20D25AE5F7._matterd._udp.local.  SRV   0 0 33333 B75AFB458ECD.local.
B75AFB458ECD.local.                 AAAA  fe80::f515:576f:9783:3f30
```

The port number 33333 is given here purely as an example.

A Commissionee can discover all Commissioners:

```
dns-sd -B _matterd._udp
```

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

or

```
avahi-browse _matterd._udp -r
```

A Commissionee can discover Commissioners with device type 35:

```
dns-sd -B _matterd._udp,_T35
```

or

```
avahi-browse _T35._sub._matterd._udp -r
```

A Commissionee can discover Commissioners with Vendor ID 123:

```
dns-sd -B _matterd._udp,_V123
```

or

```
avahi-browse _V123._sub._matterd._udp -r
```

4.3.4. Common TXT Key/Value Pairs

The TXT records provided during Commissionable, Operational and Commissioner discovery MAY contain the following optional key/value pairs which are common to every discovery method:

¥ The optional key SII indicates the SLEEPY_IDLE_INTERVAL of the Node. This key MAY optionally be provided by a Node to override sleepy defaults. If the key is not included or invalid, the Node querying the service record SHALL use the default SED parameter value. The SII value is an unsigned integer with units of milliseconds and SHALL be encoded as a variable-length decimal number in ASCII encoding, omitting any leading zeros. The SII value SHALL NOT exceed 3600000 (1 hour in milliseconds).

%Example: SII=5300 to override the initial retry interval value to 5.3 seconds.

¥ The optional key SAI indicates the SLEEPY_ACTIVE_INTERVAL of the Node. This key MAY optionally be provided by a Node to override SED defaults. If the key is not included or invalid, the Node querying the service record SHALL use the default MRP parameter value. The SAI value is an unsigned integer with units of milliseconds and SHALL be encoded as a variable-length decimal number in ASCII encoding, omitting any leading zeros. The SAI value SHALL NOT exceed 3600000 (1 hour in milliseconds).

%Example: SAI=1250 to override the active retry interval value to 1.25 seconds.

¥ The optional key T indicates whether the Node supports TCP. This key MAY optionally be provided by a Node that does not support TCP. If the key is not included or invalid, the Node query!

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

ing the service record SHALL assume the default value of T=0 indicating TCP is not supported. The T key, if included, SHALL have one of two valid values: '0' to indicate "TCP not supported", or '1' to indicate "TCP supported".

%Example: T=1 to announce TCP is supported by the advertising Node.

4.4. Message Frame Format

This section describes the encoding of the Matter message format. The Matter message format pro! vides flexible support for various communication paradigms, including unicast secure sessions, multicast group messaging, and session establishment itself. The process of encrypting Matter mes! sages is the same in all modes of communication, and assumes symmetric keys are shared between communicating parties. Unencrypted messages are used only for protocols which bootstrap secure messaging, such as session establishments.

Matter messages are used by Matter applications, as well as the Matter protocol stack itself, to con! vey application-specific data and/or commands. The Protocol portion of a Matter message contains a Protocol ID and Protocol Opcode which identify both the semantic meaning of the message as well as the structure of any associated application payload data. Matter messages also convey an Exchange ID , which relates the message to a particular exchange (i.e. conversation) taking place between two nodes. Finally, certain types of Matter messages can convey information that acknowl! edges the reception of an earlier message. This is used as part of the Message Reliability Protocol to provide guaranteed delivery of messages over unreliable transports.

All multi-byte integer fields are transmitted in little-endian byte order unless otherwise noted in the field description.

Matter messages are structured as follows:

NOTE [] denotes the field is optional.

Table 7. Matter Message format definition

Length	Field
Message Header	
2 bytes	[Message Length]
1 byte	Message Flags
2 bytes	Session ID
1 byte	Security Flags
4 bytes	Message Counter
0/8 bytes	[Source Node ID]
0/2/8 bytes	[Destination Node ID]
variable	[Message Extensions . . .]
Message Payload	
variable	[Message Payload . . .] (encrypted)

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Length	Field
Message Footer	
variable	[Message Integrity Check]

The Message Payload of a Matter message SHALL contain a Protocol Message with format as fol! lows:

Table 8. Protocol Message format definition

Length	Field
Protocol Header	
1 byte	Exchange Flags
1 byte	Protocol Opcode
2 bytes	Exchange ID
2 bytes	Protocol ID
2 bytes	[Protocol Vendor ID]
4 bytes	[Acknowledged Message Counter]
variable	[Secured Extensions . . .]
Application Payload	
variable	[Application Payload . . .]

4.4.1. Message Header Field Descriptions

4.4.1.1. Message Length (16 bits)

An optional, unsigned integer value specifying the overall length of the message in bytes, not including the size of the Message Length field itself. This field SHALL only be present when the message is being transmitted over a stream-oriented channel such as TCP. When transmitted over a message-oriented channel, the message length is conveyed by the underlying channel. For example, when transmitted over UDP, the message length is equal to the payload length of the UDP packet.

4.4.1.2. Message Flags (8 bits)

An unsigned integer bit field containing the following subfields:

Table 9. Message Flags field definition

bit 7	6	5	4	3	2	1	0
Version				-	S	DSIZ	

NOTE All unused bits in the Message Flags field are reserved and SHALL be set to zero on transmission and SHALL be silently ignored on reception.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Version (4 bits, positions 4-7)

An unsigned integer specifying the version of the Matter Message format used to encode the mes! sage. Currently only one version is defined:

¥ 0$Ñ$Matter Message Format version 1.0

¥ 1-15$Ñ$Reserved for future use

Messages with version field set to reserved values SHALL be dropped without sending a message-layer acknowledgement.

NOTE

The Version field conveys information solely about the structure of the Matter mes! sage itself, not about the structure of the application payload or the interpretation of the messageÕs type. Thus, changes to how an application handles or interprets a message do not result in the creation of a new message format version number.

S Flag (1 bit, position 2)

A single bit field which SHALL be set if and only if the Source Node ID field is present.

DSIZ Field (2 bits, position 0-1)

This field SHALL indicate the size and meaning of the Destination Node ID field.

¥ 0$Ñ$Destination Node ID field is not present

¥ 1$Ñ$Destination Node ID field is present as a 64-bit Node ID

¥ 2$Ñ$Destination Node ID field is present as a 16-bit Group ID

¥ 3$Ñ$Reserved for future use

Messages with DSIZ field set to reserved values SHALL be dropped without sending a message-layer acknowledgement.

4.4.1.3. Session ID (16 bits)

An unsigned integer value identifying the session associated with this message. The session identi! fies the particular key used to encrypt a message out of the set of available keys (either session or group), and the particular encryption/message integrity algorithm to use for the message. The Ses! sion ID field is always present. For details on derivation of this field, see respective sections on uni! cast and group session ID derivation.

4.4.1.4. Security Flags (8 bits)

An unsigned integer bit field containing the following subfields:

Table 10. Security Flags field definition

bit 7	6	5	4	3	2	1	0
P	C	MX	Reserved			Session Type	

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

NOTE	All unused bits in the Security Flags field are reserved and SHALL be set to zero on transmission and SHALL be silently ignored on reception.

P Flag (1 bit, position 7)

The Privacy (P) flag is a single bit field which, when set, SHALL identify that the message is encoded with privacy enhancements as specified in Section 4.8.3, ÒPrivacy Processing of Outgoing Mes! sagesÓ.

C Flag (1 bit, position 6)

The Control message (C) flag is a single bit field which, when set, SHALL identify that the message is a control message, such as for the Message Counter Synchronization Protocol , and uses the control message counter for the nonce field as specified in Section 4.7.1.1, ÒNonceÓ.

MX Flag (1 bit, position 5)

The Message Extensions (MX) flag is a single bit field which, when set, SHALL indicate that the Mes! sage Extensions portion of the message is present and has non-zero length. Version 1.0 Nodes SHALL set this flag to zero.

Session Type (2 bit, position 0-1)

An unsigned integer specifying the type of session associated with the message. The following val! ues are defined:

- ¥ 0$Ñ$Unicast Session
- ¥ 1$Ñ$Group Session
- ¥ 2-3$Ñ$Reserved for future use

Messages with Session Type set to reserved values are not valid and SHALL be dropped without sending a message-layer acknowledgement.

The Session Type defines how the Session ID is to be interpreted.

The Unsecured Session SHALL be indicated when both Session Type and Session ID are set to 0. The Unsecured Session SHALL have no encryption, privacy, or message integrity checking.

A Secure Unicast Session SHALL be indicated when Session Type is Unicast Session and Session ID is NOT 0.

4.4.1.5. Message Counter (32 bits)

An unsigned integer value uniquely identifying the message from the perspective of the sending node. The message counter is generated based on the Session Type and increases monotonically for each unique message generated. When messages are retransmitted, using the reliable messaging capabilities, the counter remains the same, as logical retransmission is of a given message as identi! fied by its message counter. Similarly, acknowledgements refer to values of the message counter being acknowledged.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

NOTE	The Message Counter field is scoped to a given Encryption Key. Also, the Message Counter values are independent for control messages and data messages, as indi! cated by the C Flag. So it is possible to have the same Message Counter for two mes! sages encrypted with different keys, as well as two messages encrypted with the same key but different values of the C Flag.

4.4.1.6. Source Node ID (64 bits)

An optional sequence of 8 bytes containing the unique identifier of the source node. The Source Node ID field SHALL only be present in a message when the S Flag in the Message Flags field is set to 1.

4.4.1.7. Destination Node ID

The optional Destination Node ID field contains the unique Node Identifier of the destination Node or group to which the message is being sent. The size and encoding of the Destination Node ID field depends on the value of the DSIZ field.

4.4.1.8. Message Extensions (variable)

The Message Extensions field is a variable length block of data for providing backwards compatible extensibility. The format of the Message Extensions block is shown in Table 11, ÒMessage Extensions block format definitionÓ . The Message Extensions block SHALL be present only if the MX Flag is set to 1 in the Security Flags field.

Version 1.0 Nodes SHALL ignore the contents of the Message Extensions payload.

The Message Extensions block SHALL be authenticated and privacy obfuscated based on the Secu! rity Flags settings.

Table 11. Message Extensions block format definition

Length	Field
2 bytes	Message Extensions Payload Length, in bytes
variable	[Message Extensions Payload]

4.4.2. Message Footer Field Descriptions

4.4.2.1. Message Integrity Check (variable length)

A sequence of bytes containing the message integrity check value (a.k.a. tag or MIC) for the mes! sage. The length and byte order of the field depend on the integrity check algorithm in use as speci! fied in Section 3.6, ÒData Confidentiality and IntegrityÓ .

The Message Integrity Check field SHALL be present for all messages except those of Unsecured Ses! sion Type .

The MIC is calculated as described in Section 4.7.2, ÒSecurity Processing of Outgoing MessagesÓ.

csa connectivity standards alliance Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

4.4.3. Protocol Header Field Descriptions

4.4.3.1. Exchange Flags (8 bits)

An unsigned integer bit field containing the following subfields:

Table 12. Exchange Flags field definition

bit 7	6	5	4	3	2	1	0
-	-	-	V	SX	R	A	I

NOTE All unused bits in the Exchange Flags field are reserved and SHALL be set to zero on transmission and SHALL be silently ignored on reception.

I Flag (1 bit, position 0)

The Initiator (I) flag is a single bit field which, when set, SHALL indicate that the message was sent by the initiator of the exchange.

A Flag (1 bit, position 1)

The Acknowledgement (A) flag is a single bit field which, when set, SHALL indicate that the mes! sage serves as an acknowledgement of a previous message received by the current message sender.

R Flag (1 bit, position 2)

The Reliability (R) flag is a single bit field which, when set, SHALL indicate that the message sender wishes to receive an acknowledgement for the message.

SX Flag (1 bit, position 3)

The Secured Extensions (SX) flag is a single bit field which, when set, SHALL indicate that the Secured Extensions portion of the message is present and has non-zero length. Version 1.0 Nodes SHALL set this flag to zero.

V Flag (1 bit, position 4)

The Vendor (V) protocol flag is a single bit field which, when set, SHALL indicate whether the Proto! col Vendor ID is present.

4.4.3.2. Protocol Opcode (8 bits)

An unsigned integer value identifying the type of the message. The Protocol Opcode is interpreted relative to the Matter protocol specified in the Protocol ID field.

Opcodes are defined by the corresponding Protocol specification, for example Secure Channel Pro! tocol .

4.4.3.3. Exchange ID (16 bits)

An unsigned integer value identifying the exchange to which the message belongs. An Exchange ID

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

is allocated by the initiator of the exchange, and is unique within the initiator exchange number space as specified in Section 4.9.2, ÒExchange IDÓ

4.4.3.4. Protocol ID (16 bits)

An unsigned integer value identifying the protocol in which the Protocol Opcode of the message is defined.

When the Protocol Vendor ID is the Matter Standard Vendor ID , the Protocol ID SHALL have one of the values specified by Table 13, ÒProtocol IDs for the Matter Standard Vendor IDÓ .

Table 13. Protocol IDs for the Matter Standard Vendor ID

Range	Type	Message Specification
0x0000	PROTOCOL_ID_SECURE_CHAN! NEL	Section 4.10.1, ÒSecure Channel Protocol MessagesÓ
0x0001	PROTOCOL_ID_INTERACTION_! MODEL	Section 10.2.1, ÒIM Protocol MessagesÓ
0x0002	PROTOCOL_ID_BDX	Section 11.21.3.1, ÒBDX Protocol MessagesÓ
0x0003	PROTOCOL_ID_USER_DIRECTED_ COMMISSIONING	Section 5.3.2, ÒUDC Protocol MessagesÓ
0x0004	PROTOCOL_ID_FOR_TESTING	Reserved for bespoke protocols run in an isolated test environment.
0x0005 - 0xFFFF	reserved	reserved

4.4.3.5. Protocol Vendor ID (16 bits)

An optional, unsigned integer value that contains the Vendor ID namespacing for the Protocol ID field. This field SHALL only be present when the V Flag is set; otherwise the default is 0, corre! sponding to the Matter Standard Vendor ID .

4.4.3.6. Acknowledged Message Counter (32 bits)

An optional, unsigned integer value containing the message counter of a previous message that is being acknowledged by the current message. The Acknowledged Message Counter field is SHALL only be present when the A Flag in the Exchange Flags field is 1.

4.4.3.7. Secured Extensions (variable)

The Secured Extensions field is a variable length block of data for providing backwards compatible extensibility. The format of the Secured Extensions block is shown in Table 14, ÒSecured Extensions block format definitionÓ . The Secured Extensions block SHALL be present only if the SX Flag is set to 1 in the Exchange Flags field.

Version 1.0 Nodes SHALL ignore the contents of the Secured Extensions payload.

The Secured Extensions block SHALL be encrypted and authenticated based on the Security Flags

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

settings.

Table 14. Secured Extensions block format definition

Length	Field
2 bytes	Secured Extensions Payload Length, in bytes.
variable	[Secured Extensions Payload]

4.4.3.8. Application Payload (variable length)

A sequence of zero or more bytes containing the application data conveyed by the message.

4.4.4. Message Size Requirements

Support for IPv6 fragmentation is not mandatory in Matter, and the expected supported MTU is 1280 bytes, the minimum required by IPv6. Therefore, all messages, including transport headers, SHALL fit within that minimal IPv6 MTU. This message size limit SHALL apply to the UDP transport. A message received over UDP that exceeds this message size limit SHALL NOT be processed. Mes! sages sent over TCP or BTP over BLE transports MAY exceed the message size limit if both nodes are capable of supporting larger message sizes.

4.5. Message Counters

All messages contain a 32-bit message counter assigned by the sender of the message. Message counters are assigned sequentially, by monotonically increasing the counter value maintained by the sender of the message. Message counters serve several purposes:

- ¥ Duplicate Message Detection Đ Receiving systems use message counters to detect messages that have been retransmitted by the sender, e.g. in response to packet loss in the network.
- ¥ Message Acknowledgement Đ In the Message Reliability Protocol (MRP), message counters pro! vide a way for receivers to identify messages for the purpose of acknowledging their receipt.
- ¥ Encryption Nonces Đ When encrypted messages are sent, message counters provide an encryp! tion nonce that ensures each message is encrypted in a unique manner.
- ¥ Replay Prevention Đ Related to encryption, message counters also provide a means for detect! ing and preventing the replay of encrypted messages.

4.5.1. Message Counter Types

All Nodes implement three global 32-bit counters to vend message counters for certain types of messages:

- ¥ Global Unencrypted Message Counter
- ¥ Global Group Encrypted Data Message Counter
- ¥ Global Group Encrypted Control Message Counter

Additionally, Nodes implement a separate 32-bit counter for each session as part of secure session state:

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

¥ Secure Session Message Counter

Technical details for how each counter type works are described in the following sections. Table 15, ÒMessage Counter Type OverviewÓ is provided to summarize higher-level differences between Mes! sage Counter Types:

Table 15. Message Counter Type Overview

Message Counter Type	Session Type	Lifetime	Rollover Pol! icy	Nonvolatile
Global Unencrypted	Unsecured	Unlimited	Allowed	Optional
Global Encrypted Data	Group	Operational Group Key	Allowed	Mandatory
Global Encrypted Con! trol	Group	Operational Group Key	Allowed	Mandatory
Secure Session	Unicast	Session Key	Expires	Optional

4.5.1.1. Message Counter Initialization

All message counters SHALL be initialized with a random value using the Crypto_DRBG(len = 28) + 1 primitive. Message counters are initialized to a random number to increase the difficulty of traffic analysis attacks by making it harder to determine how long a particular session has been open. The random initializer ranges from 1 to 2^{28} in order to maximize initial entropy while still reserving the vast majority of the range to actual counter values (roughly $2^{32} - 2^{28}$).

4.5.1.2. Global Unencrypted Message Counter

All Nodes SHALL implement an unencrypted message counter, which is used to generate counters for unencrypted messages.

Typically, Nodes store the Global Unencrypted Message Counter in RAM. This makes the counter sub! ject to loss whenever the system reboots or otherwise loses its state. This is permissible because retaining the Global Unencrypted Message Counter is not essential to the confidentiality or integrity of the message. In the event that the Global Unencrypted Message Counter for a Node is lost, Nodes SHALL randomize the initial value of this counter on startup per Section 4.5.1.1, ÒMessage Counter InitializationÓ .

4.5.1.3. Global Group Encrypted Message Counters

The Global Group Encrypted Message Counters are used to generate the counter for messages encrypted using group keys. There are two such counters:

¥ The Global Group Encrypted Data Message Counter is used to encode regular data messages encrypted with a group key.

¥ The Global Group Encrypted Control Message Counter is used to encode control messages encrypted with a group key.

Some Nodes might not be required to implement communication using group keys, in which case they MAY omit the Global Group Encrypted Message Counters . In contrast to the Global Unencrypted

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Message Counter, Nodes are required to persist the Global Group Encrypted Message Counters in durable storage. In particular, Nodes are required to ensure that the value of the Global Group Encrypted Message Counters never rolls back and that it is monotonic within the bounds of its range for its use for a given group key. A Node SHALL randomize the initial value of this counter on fac! tory reset per Section 4.5.1.1, ÒMessage Counter InitializationÓ .

While Global Group Encrypted Message Counters are shared by many group keys to generate nonces, rollover is not an issue as long as the Epoch Key that generates each operational group key rotates frequently enough.

NOTE If a nonce is duplicated for a given key, the security consequences are isolated only to the specific key with which the duplicate nonce occurred$Ñ$a key that has not been updated prior to rollover and has been presumably abandoned or aged out.

4.5.2. Secure Session Message Counters

A Secure Session Message Counter is a per-session, 32-bit, ephemeral counter that is used by the encoding of any encrypted messages using an associated session key. Each peer in a Secure Unicast Session SHALL maintain its own message counters, with independent counters being used for each unique session used. Session Message Counters SHALL exist for as long as the associated security session is in effect. A Node SHALL randomize the initial value of this counter on session establish! ment per Section 4.5.1.1, ÒMessage Counter InitializationÓ .

The Secure Session Message Counter history window SHALL be maintained for the lifetime of the session, and SHALL be deleted at the same time as the session keys, when the session ends.

Sessions SHALL be discarded and re-established before any Secure Session Message Counter over! flow or repetition occurs.

4.5.3. Message Counters as Encryption Nonces

In the context of encrypted messages, message counters serve as nonces for the encryption algo! rithm, ensuring that every message is encrypted in a unique manner. The uniqueness of an encrypted messageÕs counter is vital to the confidentiality of the message, as the encryption algo! rithm makes it trivial for an eavesdropper to decrypt messages if the attacker can find two different messages with the same message counter that were encrypted using the same key. Specifically, an attacker can XOR the two different messages that share the same key and nonce to obtain a "block key" which can be used to decrypt any message that uses that key and nonce.

Nodes SHOULD rotate their encryption keys on a regular basis, to ensure that old encryption keys are retired before a Global Group Encrypted Message Counter has a chance to wrap to a value previ! ously used with the encryption key. In practice, the frequency of message transmission is such that encryption keys generally rotate at a rate that is much faster than the rate at which a Global Group Encrypted Message Counter wraps. In the event that a Global Group Encrypted Message Counter wraps before the associated keys are rotated, all keys associated with that Global Group Encrypted Message Counter are considered exhausted and are no longer valid to use. In such cases, a new uni! cast session SHALL be established to the Matter Node to rotate such retired keys before secure com! munication can resume. Given the importance of confidentiality and message integrity, every effort SHOULD be made to ensure that keys are rotated on a regular basis.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

4.5.4. Replay Prevention and Duplicate Message Detection

Beyond their role as encryption nonces, message counters also serve as a means to detect repeated reception of the same message. Message duplication may occur for a number of reasons: out-of-order arrival, network latency, malicious attack, or network error. For example, a duplicate can be caused when a sender retransmits a message after failing to receive an acknowledgement, or because a malicious third party attempted to replay an old message to gain some advantage. To detect duplicate messages, Nodes maintain a history window of the message counters they have received from a particular sender (see Message Reception State). Whenever a message is received, its message counter is checked against the history window of message counters from that sender to determine whether it is a duplicate. The Message Layer SHALL discard duplicate messages before they reach the application layer.

4.5.4.1. Message Reception State

The state maintained by a Node about the messages it has received from a particular peer is referred to as *message reception state*. Nodes use this state information to determine whether a newly arrived message is a duplicate of a previous message. Message reception state is maintained on a per-peer or per-session basis, depending on the type of message encryption being used.

At a conceptual level, message reception state consists of a set of integers corresponding to the counters of all the messages that have been received from a particular peer. To limit the amount of memory required to store this state, Nodes employ a lossy compression scheme that takes advantage of the fact that message counters are generated sequentially by the sender. The scheme allows for a limited amount of out-of-order message arrivals due to network effects without inducing false detection of duplicates.

In the compressed form, message reception state is structured as a pair of values: a integer representing the largest valid, or maximum message counter received from the peer (max_message_counter), and a bitmap of size MSG_COUNTER_WINDOW_SIZE indicating which messages immediately prior to the max message have been received. The offset into the bitmap equates to the difference between the corresponding message counter and the max message counter, i.e. the first bit in the bitmap indicates whether the message with the counter of max_message_counter - 1 has been received, the second indicates whether message max_message_counter - 2 has been received, and so on. A message counter is within the range of the bitmap, also known as the message counter window, when the counter value is between [(max_message_counter - MSG_COUNTER_WINDOW_SIZE + 1) mod 2^{32}, (max_message_counter - 1) mod 2^{32}]. As messages arrive, the message reception state is queried to determine if an arriving message is new or duplicate. If a message is new, the state is then updated to reflect the arrival of the message. When a message arrives with a message counter that is logically greater than the current maximum message counter for that peer, the maximum message counter value for the peer is updated and the bitmap shifted accordingly.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

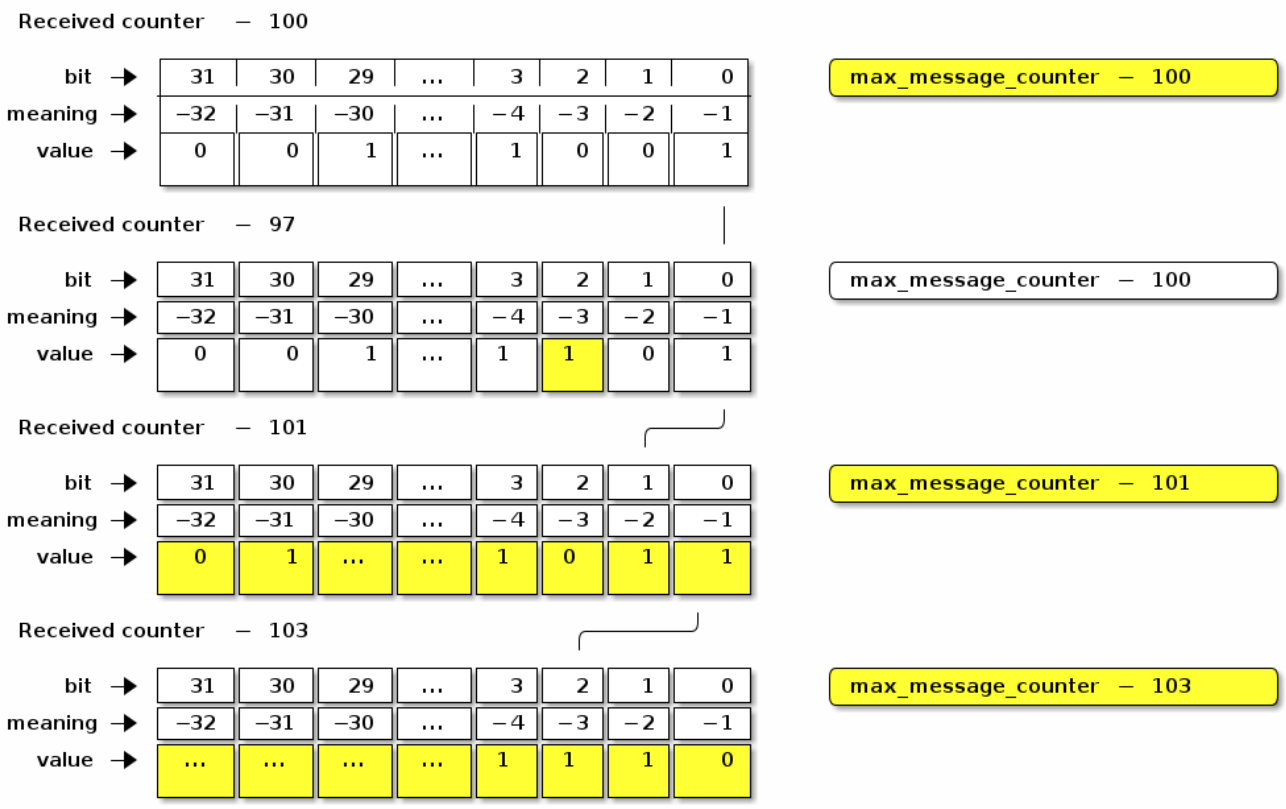


Figure 7. Message Reception State Example

4.5.4.2. Use of Message Reception State for Encrypted Messages

The algorithm for querying and updating message reception state varies slightly depending on whether the system is tracking reception of encrypted messages or unencrypted messages.

Message Counters with maximum

For encrypted messages of Secure Unicast Session Type , any arriving message with a counter in the range [(max_message_counter + 1) to (2^{32} - 1)] SHALL be considered new, and cause the max_message_counter value to be updated. Message counters within the range of the bitmap SHALL be considered duplicate if the corresponding bit offset is set to true. All other message counters SHALL be considered duplicate.

Message Counters with rollover

A message counter with rollover is a free running message counter that monotonically increases, but rolls over to zero when it exceeds the maximum value of the counter (32-bits). Group keys are secured by a shared, global message counter with rollover as described in Section 4.5.1.3, ÒGlobal Group Encrypted Message CountersÓ .

For encrypted messages of Group Session Type , any arriving message with a counter in the range [(max_message_counter + 1) to (max_message_counter + 2^{31} - 1)] (modulo 2^{32}) SHALL be considered new, and cause the max_message_counter value to be updated. Messages with counters from [(max_message_counter - 2^{31}) to (max_message_counter - MSG_COUNTER_WINDOW_SIZE) (modulo 2^{32}) SHALL be considered duplicate. A message counter equal to max_message_counter SHALL be considered duplicate. Message counters within the range of the bitmap SHALL be considered duplicate

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

if the corresponding bit offset is set to true.

This scheme for encrypted messages effectively divides the message counter space in half: those counters that are forward of the max message counter, which are considered new, and those counters that are behind the max message counter, which are considered duplicates unless indicated otherwise by the values in the bitmap.

4.5.4.3. Use of Message Reception State for Unencrypted Messages

For unencrypted messages, the algorithms for tracking messages and detecting duplicates are similar to, but more permissive than for encrypted messages using Section 4.5.4.2.2, ÒMessage Counters with rolloverÓ . This reflects the fact that duplicate detection of unencrypted messages is not done for security reasons, but rather to catch duplicates caused by network errors (e.g. loss of an ack), which are generally more bounded in time. The more relaxed algorithm for unencrypted duplicate detection also relaxes the durability requirement on the senderÕs message counter, allowing senders to store the counter in RAM.

For unencrypted messages, any message counter equal to max_message_counter or within the message counter window, where the corresponding bit is set to true SHALL be considered duplicate. All other message counters, whether behind the window or ahead of max_message_counter, are considered new and shall update max_message_counter and shift the window accordingly. Messages with a counter behind the window are likely caused by a node rebooting and are thus processed as rolling back the window to the current location. Note that when max_message_counter is close to the minimum of the range, the window shall roll back to cover message counters near the maximum of the range.

4.5.4.4. Message Reception State Initialization

To initialize Message Reception State for a given Peer Node ID, initial max_message_counter, Message Type (control or data), Encryption Level (encrypted or unencrypted), and Encryption Key (for any Encryption Level except unencrypted):

- The Message Reception State fields SHALL be set as follows:
  - The Peer Node ID SHALL reference the given Peer Node ID.
  - The Message Type SHALL be the given Message Type.
  - The Encryption Level SHALL be the given Encryption Level.
  - If the Encryption Level is NOT unencrypted, the Encryption Key SHALL reference the given key.
  - The max_message_counter SHALL be set to the given max_message_counter
  - The Message Counter bitmap SHALL be set to all 1, indicating that only new messages with counter greater than max_message_counter SHALL be accepted.

4.5.5. Counter Processing of Outgoing Messages

1. Obtain the outgoing message counter of the sending Node for the given Security Flags, Session Id, and Encryption Key:

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

a. A message of Unsecured Session Type SHALL use the current Global Unencrypted Message Counter.

b. A message of Secure Unicast Session Type SHALL use the current Secure Session Message Counter for the session associated with the Session ID.

c. A message of Group Session Type SHALL use:

i. The Global Group Encrypted Data Message Counter if the Security Flags C Flag = 0.

ii. The Global Group Encrypted Control Message Counter if the Security Flags C Flag = 1.

2. The outgoing message counter from step 1 SHALL be incremented by 1.

3. Store the incremented outgoing message counter in the OutgoingMessageCounter element asso! ciated with the Session Context for the message.

a. If the message counter wraps around from 0xFFFF_FFFF to 0x0000_0000 and the message is of Secure Unicast Session Type:

i. The Encryption Key SHALL be expired in the Session Context. The session will need to be renegotiated to resume communication after transmission of this final message.

4.5.6. Counter Processing of Incoming Messages

1. Determine the Message Reception State for the sending peer and get the current max_message_! counter.

a. Given a decrypted message of Unicast Session Type:

i. Get the session-specific Message Reception State from the Secure Unicast Session Con! text.

b. Given a decrypted message of Group Session Type:

i. Extract the Source Node ID from the Message Header.

A. If there is no Source Node ID for the message, drop the message.

ii. Get the Message Reception State for the Source Node ID of the given message:

A. If the Security Flags C Flag = 0, get the Data Message Reception State for the peer node.

B. If the Security Flags C Flag = 1, get the Control Message Reception State for the peer node.

iii. If there is no Message Reception State for the groupcast message, initiate Section 4.16.4, ÒUnsynchronized Message ProcessingÓ.

c. Given an unencrypted message:

i. Get the Message Reception State associated with the Unsecured Session Context.

ii. If there is no Message Reception State for the unencrypted message, create it with the information from the given message.

2. If the Message Counter is outside the valid message counter window, the message SHALL be marked as a duplicate. Note that while messages may be outside of the window for reasons other than being a duplicate, and we always mark them as such.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

3. If the message is a duplicate:
   a. If the message is marked as encrypted, follow Section 4.5.4.2, ÒUse of Message Reception State for Encrypted MessagesÓ .
   b. If the message is marked as unencrypted, follow Section 4.5.4.3, ÒUse of Message Reception State for Unencrypted MessagesÓ .
   c. If the message is encrypted and marked as a duplicate, i.e. Message Counter is outside the valid message counter window or marked as previously received in the Message Reception State:
      i. Perform Section 4.11.5.2, ÒReliable Message Processing of Incoming MessagesÓ on the duplicate message.
   d. Otherwise, update the Message Reception State as detailed in Section 4.5.4.1, ÒMessage Reception StateÓ, and accept the message for further processing.

4.6. Message Processing

This sub-clause describes the fundamental procedures for transmission and reception.

4.6.1. Message Transmission

To prepare a message for transmission with a given Session ID, Destination Node ID (which may be a group node id or an operational node id) and Security Flags, the following steps SHALL be per! formed, in order:

1. Process the message as described in Section 4.5.5, ÒCounter Processing of Outgoing MessagesÓ.
2. If the messageÕsSession Type is a Unicast Session :
   a. Set SessionTimestampto a timestamp from a clock which would allow for the eventual deter! mination of the last session use relative to other sessions.
   b. Process the message as described in Section 4.7.2, ÒSecurity Processing of Outgoing Mes! sagesÓ.
   c. Process the message as described in Section 4.8.3, ÒPrivacy Processing of Outgoing Mes! sagesÓ.

4.6.2. Message Reception

To process a received message, the following steps SHALL be performed in order:

1. Perform validity checks on the message; if any fail, processing of the message SHALL stop, and a 'message invalid' error SHOULD be indicated to the next higher layer:
   a. The Version field SHALL be 0.
   b. If the message is of Secure Unicast Session Type :
      i. The DSIZ field SHALL NOT indicate a Group ID is present.
   c. If the message is of Group Session Type :
      i. The DSIZ field SHALL NOT be 0.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

ii. The S Flag field SHALL NOT be 0.

2. If the message is NOT of Unsecured Session Type :
   a. Obtain the Privacy and Encryption Keys associated with the given Session ID:
      i. If no keys are found, security processing SHALL indicate a failure to the next higher layer with a status of 'message security failed' and no further security processing SHALL be done on this message.
   b. For each Privacy and Encryption Key, of which there may be more than one in the case of group messages:
      i. If the P Flag is set, follow Section 4.8.4, ÒPrivacy Processing of Incoming MessagesÓ to deobfuscate the message.
      ii. Follow Section 4.7.3, ÒSecurity Processing of Incoming MessagesÓ to decrypt and authen! ticate the message.
3. Follow Section 4.5.6, ÒCounter Processing of Incoming MessagesÓ to enforce replay protection and duplicate detection.
4. If the message transport is UDP, follow Section 4.11.5.2, ÒReliable Message Processing of Incom! ing MessagesÓto process message reliability.
5. If the messageÕsSession Type is a Unicast Session :
   a. Set SessionTimestamp and ActiveTimestamp to a timestamp from a clock which would allow for the eventual determination of the last session use relative to other sessions.
6. The received message is then delivered to Section 4.9.5, ÒExchange Message ProcessingÓ

4.7. Message Security

The detailed steps involved in security processing of outgoing and incoming Matter messages are described in Section 4.7.2, ÒSecurity Processing of Outgoing MessagesÓ and Section 4.7.3, ÒSecurity Processing of Incoming MessagesÓ respectively. Section 4.7.1, ÒData confidentiality and integrity with data origin authentication parametersÓ defines how the cryptographic parameters are set for securing Matter messages.

4.7.1. Data confidentiality and integrity with data origin authentication parameters

This section specifies the parameters to use the data confidentiality and integrity cryptographic primitive as defined in Section 3.6, ÒData Confidentiality and IntegrityÓ .

The parameters in this section SHALL apply for all encrypted messages, i.e. all messages except those of Unsecured Session Type .

4.7.1.1. Nonce

The nonce SHALL be formatted as specified in Table 16, ÒNonceÓ.

Table 16. Nonce

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Octets: 1	4	8
Security Flags	Message Counter	Source Node ID

The nonce used for the Authenticated Encryption with Additional Data (AEAD) algorithm (see Section 3.6, ÒData Confidentiality and IntegrityÓ) for a given message SHALL be defined as the concatenation of the Security Flags, the Message Counter, and the Source Node ID of that message. The scalar fields in the nonce, namely the Message Counter and the Source Node ID SHALL be encoded in little-endian byte order for the purposes of serialization within the nonce, that is, in the same byte ordering as the segment of the message from which its data originates.

The Source Node ID field used in the nonce SHALL be set to the Operational Node ID of the node originating security protection of the message:

¥ If the message is of Secure Unicast Session Type :

%For a CASE session, the Nonce Source Node ID SHALL be determined via the Secure Session Context associated with the Session Identifier.

%For a PASE session, the Nonce Source Node ID SHALL be Unspecified Node ID .

¥ If the message is of Group Session Type :

%The S Flag of the message SHALL be 1 and the Nonce Source Node ID SHALL be the Source Node ID of the message.

%If the S Flag of the message is 0 the message SHALL be dropped.

NOTE Because PASE negotiates strong one-time keys per session and the I2RKey and R2IKey are distinct for each direction of communication, the use of the Unspecified Node ID as the Nonce Source Node ID remains semantically secure.

4.7.2. Security Processing of Outgoing Messages

The process for encrypting Matter messages is depicted graphically in Figure 8, ÒMatter Message EncryptionÓ with color code conventions described in Figure 9, ÒMatter Message Encryption LegendÓ.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

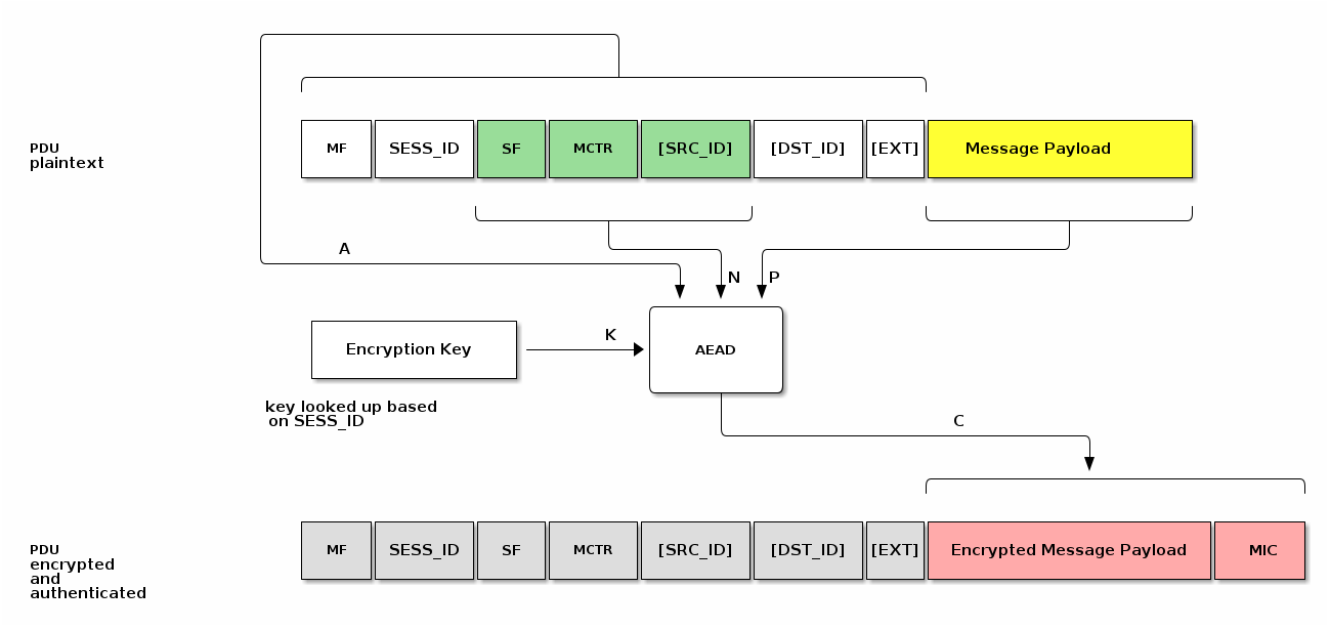


Figure 8. Matter Message Encryption

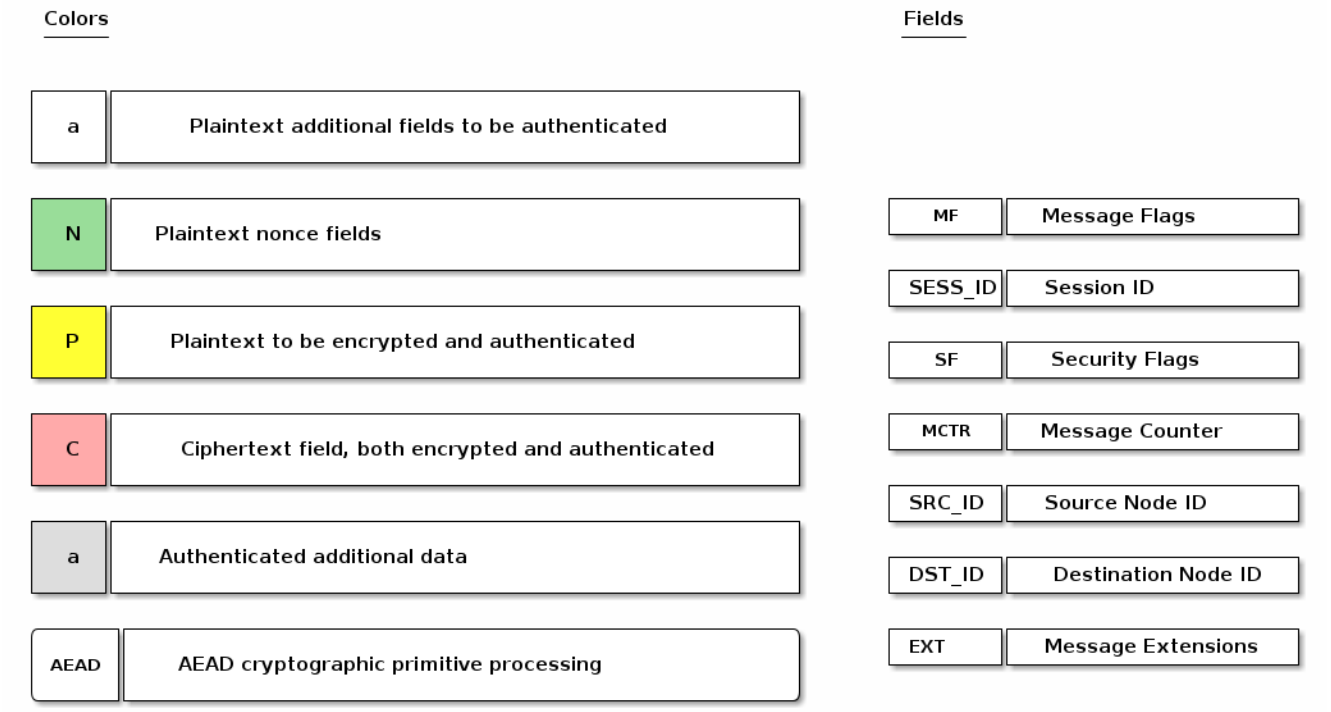


Figure 9. Matter Message Encryption Legend

To prepare a secure message for transmission with a given Session ID, Destination Node ID (which may be a group node id or an operational node id) and Security Flags, the Node SHALL perform the following steps:

1. Obtain the Encryption Key associated with the Session ID and Destination Node ID and the Ses! sion Type associated with the Destination Node ID:
   a. If no key is found for the given Session ID, security processing SHALL fail and no further security processing SHALL be done on this message.
2. Obtain the outgoing message counter of the sending Node as per Section 4.5.5, ÒCounter Process!

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

ing of Outgoing MessagesÓ.

3. Set the Security fields as follows:
   a. The Session ID field SHALL be set to the value provided to step 1.
   b. The Security Flags field SHALL be set to the value provided to step 1 with the following sub! fields updated:
      i. The Session Type field SHALL be set to the value obtained from step 1.
4. Set the Message Flags, Destination Node ID , and Source Node ID fields as follows:
   a. If the Session Type is a unicast session:
      i. Set S Flag to 0.
      ii. Set DSIZ to 0.
      iii. Omit both Destination Node ID , and Source Node ID .
   b. Else if the Session Type is a group session:
      i. Set S Flag to 1.
      ii. Set DSIZ to 2.
      iii. Set Source Node ID field to the operational node id of the sending node.
      iv. Set Destination Node ID field to the 16-bit Group ID derived from the Destination Node ID.
5. Set the Message Counter field to the outgoing message counter from step 2.
6. Execute the AEAD generate and encrypt operation, as specified in Section 3.6.1, ÒGenerate and encryptÓ, with the following instantiations:
   a. The bit string key K SHALL be the Encryption Key obtained from step 1;
   b. The nonce N SHALL be the CRYPTO_AEAD_NONCE_LENGTH_BYTES octet string constructed accord! ing to Table 16, ÒNonceÓ;
   c. The parameter P SHALL be the Message Payload;
   d. The additional data octet string A SHALL be the Message Header contents, using little-endian byte order for all scalars, exactly as they appear in the message segments from which they originate:

      ```
      Message Flags || Session ID || Security Flags || Message Counter
      ```

      with the optional fields appended according to the Message Flags:

      ```
      [Source Node ID] || [Destination Node ID] || [Message Extensions]
      ```

   e. C = Crypto_AEAD_GenerateEncrypt(K, P, A, N)
7. If the AEAD operation invoked in step 6 results in an error, then security processing SHALL fail and no further security processing SHALL be done on this message.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

8. Let C be the output from step 6. C contains the tag of CRYPTO_AEAD_MIC_LENGTH_BITS bits (Message Integrity Check (MIC)) as specified by Section 3.6.1, ÒGenerate and encryptÓ. The secured outgoing message SHALL be:

```
A || C
```

4.7.3. Security Processing of Incoming Messages

All incoming message processing SHALL occur in a serialized manner. If an implementation chooses to process messages in a parallel manner, it must ensure that the behavior is opaque-box identical to a serialized processing implementation.

If the transport layer receives a secured message as indicated by the Session ID, it SHALL perform the following steps:

1. Determine the Session Type, Session ID, and Message Counter from the message header of the received message.
2. Obtain the Encryption Key associated with the Session Context of the given Session ID and Session Type:
   a. If no key is found for the given Session ID, security processing SHALL indicate a failure to the next higher layer with a status of 'message security failed' and no further security processing SHALL be done on this message.
3. Execute the AEAD decryption and verification operation as specified in Section 3.6.2, ÒDecrypt and verifyÓ with the following instantiations:
   a. The bit string key K SHALL be the Encryption Key obtained from step 2;
   b. The nonce N SHALL be the CRYPTO_AEAD_NONCE_LENGTH_BYTES octet string constructed according to Table 16, ÒNonceÓ;
   c. The parameter C SHALL be the encrypted and authenticated Message Payload;
   d. The additional data octet string A SHALL be the authenticated Message Header:
   e. {success, P} = Crypto_AEAD_DecryptVerify(K, C, A, N)
4. Return the result {success, P} of the AEAD operation:
   a. If the success is FALSE, security processing SHALL fail and further processing SHALL NOT be performed on this message. An appropriate error SHOULD be raised to the upper layer to indicate the error.
   b. Otherwise, set the octet string PlaintextMessage to the string

```
A || P
```

5. PlaintextMessage now represents the deciphered, authenticated, received message.
   a. NOTE: The message has not yet undergone counter processing nor replay detection.
   b. The PlaintextMessage SHALL be marked as successfully security processed and SHALL be

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

released to the next processing layer.

4.8. Message Privacy

Privacy processing of a message describes the obfuscation and deobfuscation of the message header fields after encryption and before decryption.

The detailed steps involved in privacy processing of outgoing and incoming Matter messages are described in Section 4.8.3, ÒPrivacy Processing of Outgoing MessagesÓ and Section 4.8.4, ÒPrivacy Processing of Incoming MessagesÓ respectively. They rely on the cryptographic primitives in Section 3.7, ÒMessage privacyÓ.

4.8.1. Privacy Key

The Privacy Key is a symmetric key specifically used for Privacy Processing that is derived from the EncryptionKey used for Security Processing of a given message. Given a Session ID reference to a specific Encryption Key, the Privacy Key is derived as follows:

```
PrivacyKey =
Ê      Crypto_KDF
Ê      (
Ê        InputKey = EncryptionKey,
Ê        Salt = [],
Ê        Info = "PrivacyKey",
Ê        Length = CRYPTO_SYMMETRIC_KEY_LENGTH_BITS
Ê      )
```

4.8.2. Privacy Nonce

The Privacy Nonce is a nonce specifically used for Privacy Processing that is derived from the SessionId and MIC of the message. The Privacy Nonce SHALL be the CRYPTO_AEAD_NONCE_LENGTH_BYTES-octet string constructed as the 16-bit Session ID (in big-endian format) concatenated with the lower 11 (i.e. CRYPTO_AEAD_MIC_LENGTH_BYTES-5) bytes of the MIC:

```
Ê  PrivacyNonce = Session ID || MIC[5..15]
```

For example if Session ID is 42 (i.e. 0x002A) and the computed MIC is c5:a0:06:3a:d5:d2:51:81:91:40:0d:d6:8c:5c:16:3b :

```
Ê  Session ID = 00:2a
Ê  MIC = c5:a0:06:3a:d5:d2:51:81:91:40:0d:d6:8c:5c:16:3b
Ê  MIC[5..15] = d2:51:81:91:40:0d:d6:8c:5c:16:3b
Ê  PrivacyNonce = SessionID || MIC[5..15] = 00:2a || d2:51:81:91:40:0d:d6:8c:5c:16:3b
Ê  PrivacyNonce = 00:2a:d2:51:81:91:40:0d:d6:8c:5c:16:3b
```

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

4.8.3. Privacy Processing of Outgoing Messages

The process for privacy encoding Matter message headers is depicted graphically in Figure 10, ÒMatter Message PrivacyÓ.

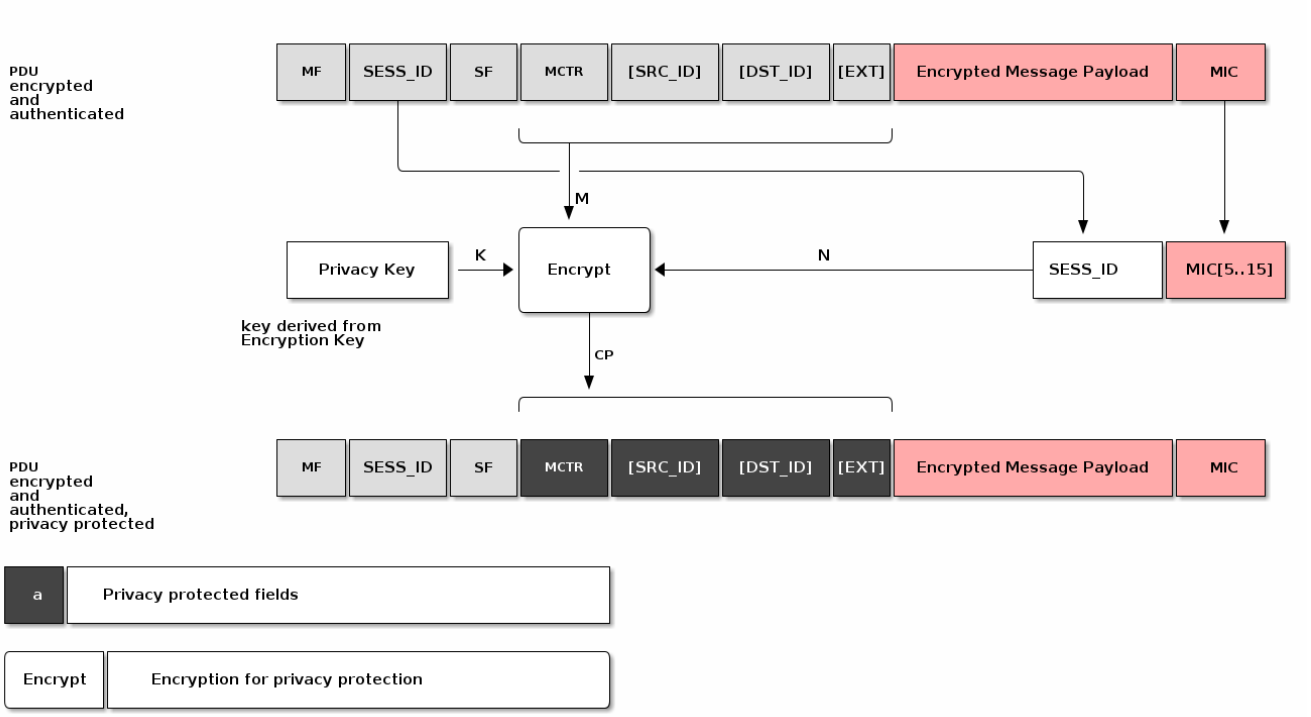


Figure 10. Matter Message Privacy

To apply privacy obfuscation to an encrypted message prepared for transmission by Section 4.6.1, ÒMessage TransmissionÓ, apply obfuscation steps as follows:

1. If P Flag is not set, do nothing.
2. Obtain the Privacy Key for the Encryption Key used to secure the message.
3. Execute the encryption operation as specified in Section 3.7.1, ÒPrivacy encryptionÓ with the fol! lowing instantiations:
   a. The bit string key K SHALL be the Privacy Key obtained from step 1;
   b. The MIC SHALL be the last CRYPTO_AEAD_MIC_LENGTH_BYTES bytes of the C outcome of the mes! sage security protection as specified in Section 4.7.2, ÒSecurity Processing of Outgoing Mes! sagesÓ (MIC = C[(CRYPTO_AEAD_MIC_LENGTH_BYTES-1)..0]
   c. The nonce N SHALL be the PrivacyNonce of the message.
   d. The parameter M SHALL be the message header fields where optional fields are only present in M if they are present in the message:

      M = Message Counter || [Source ID] || [Destination ID] || [Message Extensions]

   e. CP = Crypto_Privacy_Encrypt(K, M, N)
4. Let CP be the obfuscated output from step 2. CP SHALL be used in the final private message in place of the message header fields.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

4.8.4. Privacy Processing of Incoming Messages

To deobfuscate a private message received by Section 4.6.2, ÒMessage ReceptionÓ with a given Privacy Key, perform security processing as follows:

1. If P Flag is not set, do nothing.
2. With the given Privacy Key, execute the decryption as specified in Section 3.7.2, ÒPrivacy decryptionÓ with the following instantiations:
   a. The bit string key K SHALL be the Privacy Key obtained from step 1;
   b. The MIC SHALL be the last CRYPTO_AEAD_MIC_LENGTH_BYTES bytes of the C outcome of the message security protection as specified in Section 4.7.3, ÒSecurity Processing of Incoming MessagesÓ (MIC = C[(CRYPTO_AEAD_MIC_LENGTH_BYTES-1)..0])
   c. The nonce N SHALL be the PrivacyNonce of the message.
   d. The parameter CP SHALL be the message header fields where optional fields are only present in CP if they are present in the message:

      ```
      CP = Message Counter || [Source ID] || [Destination ID] || [Message Extensions]
      ```

   e. M = Crypto_Privacy_Decrypt(K, CP, N)
3. Let M be the deobfuscated output from step 2.
   a. M SHALL be used in the final private message in place of the message header fields.
   b. The first successfully validated message, M, by Section 4.7.3, ÒSecurity Processing of Incoming MessagesÓ SHALL terminate iteration through Privacy Keys in step 2.

4.9. Message Exchanges

An Exchange provides a way to group related messages together, organize communication flows, and enable higher levels of the communication stack to track semantically relevant groupings of messages.

An Exchange SHALL be bound to exactly one underlying session that will transport all associated Exchange messages for the life of that Exchange. The underlying session SHALL be one of the following session types: secure unicast (as established by PASE or CASE), unsecured (as is used for the initial session establishment phase of a PASE/CASE session), secure group, or MCSP.

When used with reliability, Exchanges assume basic flow control by the upper layer. The Exchange Layer SHALL not accept a message from the upper layer when there is an outbound reliable message pending on the same Exchange.

4.9.1. Exchange Role

The first Node to send a message in an Exchange is said to be in the Initiator role, and all the other Nodes that subsequently participate in the Exchange are said to be in a Responder role. An Exchange is always between one Initiator and one or more peer Responder Nodes. An Exchange

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

does not survive a reboot of one of the participants. Adjacent layers MAY close an Exchange at any time.

4.9.2. Exchange ID

An Exchange of messages is identified by the Exchange ID field described in Section 4.4.3.3, ÒExchange ID (16 bits)Ó. The Exchange ID is allocated by the Initiator. The first message the Initiator sends in a new Exchange SHALL contain a fresh value for the Exchange ID field. The Exchange is then identified by the tuple { Session Context , Exchange ID, Exchange Role} where Session Context is one of an Unsecured , Secured , Groupcast or MCSP session context. All messages that are part of a given Exchange, whether they are sent by the Initiator or not, share the same Exchange ID, allowing the Initiator and Responder Nodes to match responses to requests or otherwise group messages together that are part of more complex transactions. The first Exchange ID for a given Initiator Node SHALL be a random integer. All subsequent Exchange IDs created by that Initiator SHALL be the last Exchange ID it created incremented by one. An Exchange ID is an unsigned integer that rolls over to zero when its maximum value is exceeded.

4.9.3. Exchange Context

An Exchange context is the metadata tracked for an Exchange by all exchange participants. An Exchange context tracks the following data:

1. Exchange ID: The Exchange ID assigned by the Initiator
2. Exchange Role: Initiator or Responder
3. Session Context : The underlying Unsecured , Secured , Groupcast or MCSP session context

%Together, Session Context, Exchange ID and Role comprise a unique key allowing participants to identify any exchange.

4.9.3.1. Protocol ID Registration

The Interaction Model layer indicates to the Exchange Layer which Protocols it will accept. Any message for a Protocol ID that is not registered with the Exchange Layer SHALL be dropped.

4.9.4. Exchange Message Dispatch

When sending a message to the Exchange Layer, the next higher layer SHALL specify whether the message is part of an existing Exchange, or the first of a new Exchange. For the case of a first message, the Initiator creates a new Exchange. The Node in the Initiator role SHALL always set the I Flag in the Exchange Flags of every message it sends in that Exchange.

Each Node in a Responder role for an Exchange SHALL use the Exchange ID received in previous messages for the Exchange. Each Node in the Responder role SHALL NOT set the I Flag in the Exchange Flags of every message it sends in that Exchange. Each Node in a Responder role SHALL NOT set the Destination Node ID field to a value that identifies any Node other than the Node in the Initiator role for the Exchange.

Processing SHALL then proceed to Section 4.11.5.1, ÒReliable Message Processing of Outgoing MessagesÓ.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

4.9.5. Exchange Message Processing

After completion of Section 4.6.2, ÒMessage ReceptionÓ, if the message matches an existing Exchange, it is dispatched to the appropriate protocol handler in the next higher layer. Messages for an existing Exchange are dispatched to the handler for that Exchange. Otherwise, the unsolicited message that created the Exchange is dispatched to the unsolicited message handler.

4.9.5.1. Exchange Message Matching

Upon receipt of a message, the Exchange Layer attempts to match the message to an existing Exchange. A given message is part of an Exchange if it satisfies all the following criteria:

1. The message was received over the session associated with the Exchange.
2. The Exchange ID of the message matches the Exchange ID of the Exchange,
3. The message has the I Flag set and the Exchange Role of the Exchange is Responder,
   OR the message does not have the I Flag set and the Exchange Role of the Exchange is Initiator.

If the message does not match an existing Exchange, the message is considered an unsolicited message.

4.9.5.2. Unsolicited Message Processing

An unsolicited message is processed as follows:

1. If the unsolicited message is not marked as having a duplicate message counter, has a registered Protocol ID, and the I Flag is set:
   a. Create a new exchange from the incoming message.
   b. The new exchange will be used by the upper layer for generating responses and subsequent processing of the message.
2. Otherwise, if the message has the R Flag set:
   a. Create an ephemeral exchange from the incoming message and send an immediate standalone acknowledgement.
   b. The message SHALL NOT be forwarded to the upper layer, and excluding the sending of an immediate standalone acknowledgment, SHALL be ignored.
   c. The ephemeral exchange created for such duplicate or unknown messages with R Flag set is automatically closed in Section 4.11.5.2.2, ÒStandalone acknowledgement processingÓ.
3. Otherwise, processing of the message SHALL stop.

Creating an Exchange based on an Incoming Message

The steps to create a new Exchange based on an incoming message are as follows:

1. A new Exchange and Exchange Context SHALL be created with the following settings:
   a. The Exchange ID SHALL be set to the Exchange ID of the message.
   b. The Exchange Role SHALL be set to the inverse of the incoming message I Flag, for example set the Exchange Role to Responder if the message is from an Initiator.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

c. The Session Context SHALL be set to the Session on which the message was received.

A node SHOULD limit itself to a maximum of 5 concurrent exchanges over a unicast session. This is to prevent a node from exhausting the message counter window of the peer node.

4.9.5.3. Closing an Exchange

An Exchange MAY be closed by the application layer or a fatal connection error from the lower message layer. The process of closing an Exchange follows:

1. Any pending acknowledgements associated with the Exchange SHALL be flushed. If there is a pending acknowledgment in the acknowledgement table for the Exchange and it has StandaloneAckSent set to false:
   a. Immediately send a standalone acknowledgement for the pending acknowledgement.
   b. Remove the acknowledgement table entry for the pending acknowledgement.
2. Wait for all pending retransmissions associated with the Exchange to complete.
   a. If the retransmission list for the Exchange is empty, remove the Exchange.
   b. Otherwise, leave the Exchange open and only close it once the retransmission list is empty.

These steps are depicted in Figure 11, ÒExchange close flowÓ.

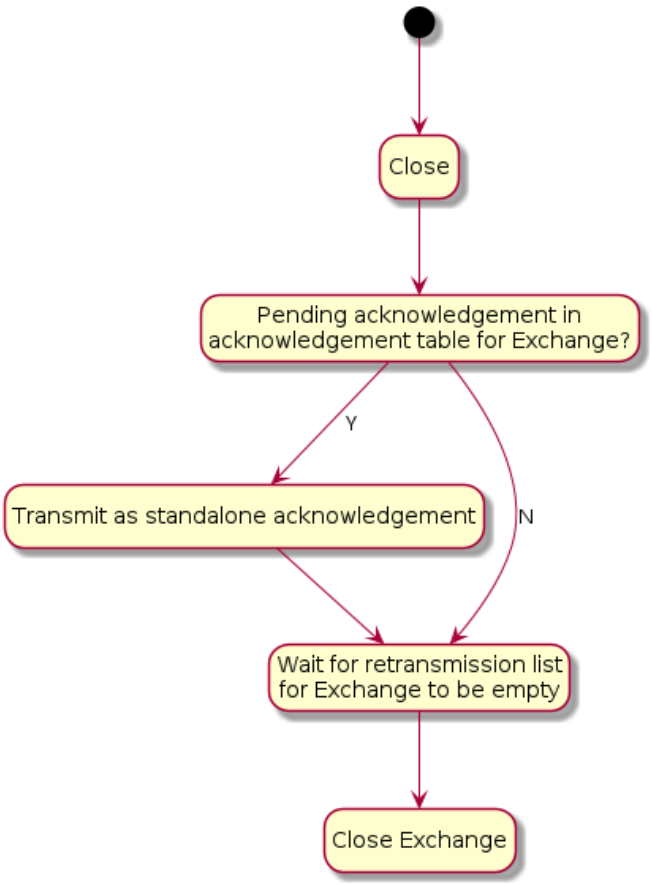


Figure 11. Exchange close flow

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

4.10. Secure Channel Protocol

This section specifies the formal protocol definition for the Secure Channel Protocol. Secure Chan! nel Protocol defines the control plane for secure channel communication and security.

4.10.1. Secure Channel Protocol Messages

Secure Channel Protocol is composed of a collection of sub-protocols, including:

- ¥ Message Counter Synchronization Protocol (MCSP)
- ¥ Message Reliability Protocol (MRP)
- ¥ Passcode Based Session Establishment (PASE)
- ¥ Certificate Based Session Establishment (CASE)

The protocol opcodes for messages within the Secure Channel Protocol are grouped based on the underlying sub-protocol that uses the message type. Table 17, ÒSecure Channel Protocol OpcodesÓ lists the messages defined by Secure Channel Protocol.

Table 17. Secure Channel Protocol Opcodes

Protocol Opcode	Protocol Command Name	Description
Protocol ID = PROTOCOL_ID_SECURE_CHANNEL		
0x00	MsgCounterSyncReq	The Message Counter Synchronization Request message queries the current message counter from a peer to boot! strap replay protection.
0x01	MsgCounterSyncRsp	The Message Counter Synchronization Response message provides the current message counter from a peer to boot! strap replay protection.
0x10	MRP Standalone Acknowl! edgement	This message is dedicated for the purpose of sending a stand-alone acknowledgement when there is no other data message available to piggyback an acknowledgement on top of.
0x20	PBKDFParamRequest	The request for PBKDF parameters necessary to complete the PASE protocol.
0x21	PBKDFParamResponse	The PBKDF parameters sent in response to PBKDF! ParamRequest during the PASE protocol.
0x22	PASE Pake1	The first PAKE message of the PASE protocol.
0x23	PASE Pake2	The second PAKE message of the PASE protocol.
0x24	PASE Pake3	The third PAKE message of the PASE protocol.
0x30	CASE Sigma1	The first message of the CASE protocol.
0x31	CASE Sigma2	The second message of the CASE protocol.
0x32	CASE Sigma3	The third message of the CASE protocol.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Protocol Opcode	Protocol Command Name	Description
0x33	CASE Sigma2_Resume	The second resumption message of the CASE protocol.
0x40	StatusReport	The Status Report message encodes the result of an opera! tion in the Secure Channel as well as other protocols.

4.10.1.1. Session Establishment - Out of Resources

After a successful session establishment using CASE or PASE, a responder may not have enough resources to save all of the session context information . To free resources, a responder SHALL evict an existing session using the following procedure:

1. Use the SessionTimestamp to determine the least-recently used session.
2. Determine the session that was least-recently used then:
   a. Send a status report : StatusReport(GeneralCode: SUCCESS, ProtocolId: SECURE_CHANNEL, Pro! tocolCode: CLOSE_SESSION) message to the peer node
   b. Remove all state associated with the session (see Section 4.12.2.1, ÒSecure Session ContextÓ). The Node MAY save state necessary to perform Session Resumption , see Section 4.13.2.2.1, ÒSession Resumption StateÓ for more details.
3. Respond to the initiator with the appropriate session establishment message

4.10.1.2. Status Report

The Status Report message is sent from protocol handlers to convey the status of an operation using a common format as defined in Appendix D, Status Report Messages. The StatusReport message is a part of the Secure Channel protocol, but embeds an additional context-specific ProtocolID field in its message-specific payload. In this way, the StatusReport can convey status for any protocol han! dler.

4.10.1.3. Secure Channel Status Report Messages

Status Reports specific to the Secure Channel are designated by embedding the PROTOCOL_ID_SE! CURE_CHANNEL in the ProtocolId field of the StatusReport body. All Secure Channel Status Report Mes! sages SHALL use the PROTOCOL_ID_SECURE_CHANNEL ProtocolId. For example, a failure to find a com! mon root of trust may be written in the specification as follows: StatusReport(GeneralCode: FAILURE, ProtocolId: SECURE_CHANNEL, ProtocolCode: NO_SHARED_TRUST_ROOTS).

There are several cases for which the secure channel layer may emit a status report:

1. To indicate successful session establishment
2. In response to errors during session establishment
3. In response to errors after session establishment
4. To indicate that a Node is terminating a session

For each of these cases, a Secure Channel Status Report message SHALL be sent with an appropriate ProtocolCode as detailed below.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

The following table describes the Secure Channel Status Report Protocol Specific codes. Each entry in the list details the appropriate General Code to be utilized with the message and whether it may be sent unencrypted. Secure Channel Status Report messages which are marked as encrypted below SHALL only be sent encrypted in a session established with CASE or PASE.

Table 18. Secure Channel Protocol Codes

Protocol Code	Error	General Code	Encrypted	Additional Data	Description
0x0000	SESSION_ESTABLISHMENT_SUCCESS	SUCCESS	N	N	Indication that the last session establishment message was successfully processed.
0x0001	NO_SHARED_TRUST_ROOTS	FAILURE	N	N	Failure to find a common set of shared roots.
0x0002	INVALID_PARAMETER	FAILURE	N	N	Generic failure during session establishment.
0x0003	CLOSE_SESSION	SUCCESS	Y	N	Indication that the sender will close the current session. See Section 4.10.1.4, ÒCloseSessionÓ for more details.
0x0004	BUSY	BUSY	N	Y	Indication that the sender can! not currently fulfill the request. See Section 4.10.1.5, ÒBusyÓ for more details.

4.10.1.4. CloseSession

A node may choose to close a session for a variety of reasons including, but not limited to, the fol! lowing:

1. The interaction between nodes is complete
2. The node needs to free up resources for a new session
3. Fabric configuration associated with the CASE session was removed with the RemoveFabric command invoked by an Administrator while the session was open

The CloseSession StatusReport SHALL only be sent encrypted within an exchange associated with a PASE or CASE session. The CloseSession StatusReport SHALL be sent within a new exchange and SHALL NOT set the R Flag.

If a Node has either sent or received a CloseSession StatusReport, that Node SHALL remove all state associated with the session (see Section 4.12.2.1, ÒSecure Session ContextÓ). The Node MAY save state necessary to perform Session Resumption , see Section 4.13.2.2.1, ÒSession Resumption StateÓ for more details.

4.10.1.5. Busy

When a receiver receives a request to start a new secure session via a Sigma1 or PBKDFParamRe!

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

quest message, the receiver MAY respond with the BUSY StatusReport when it is unable to fulfill the request. The BUSY StatusReport SHALL:

1. Set the R Flag to 0
2. Set the S Flag to 0
3. Set the StatusReport ProtocolData to a 16-bit (two byte) little-endian value indicating the mini! mum time in milliseconds to wait before retrying the original request.
4. Set the Exchange ID to the Exchange ID present in the Sigma1 or PBKDFParamRequest message which triggered this response.

For example, a responder wishing to indicate they are unable to fulfill the request and that the ini! tiator should wait 500 milliseconds before trying again would send StatusReport(GeneralCode: BUSY, ProtocolId: SECURE_CHANNEL, ProtocolCode: BUSY, ProtocolData: [0xF4, 0x01]) .

The BUSY StatusReport SHALL NOT be sent in response to any message except for Sigma1 or PBKDFParamRequest .

An initiator receiving a BUSY StatusReport from a responder SHALL wait for at least a period of t milliseconds before retrying the request where t is the value obtained from the Busy StatusReport ProtocolData field.

If the initiator sends a new session establishment request after receiving a BUSY StatusReport, the request SHALL contain new values for all randomized parameters.

4.10.2. Parameters and Constants

Table 19, ÒGlossary of constantsÓ is a glossary of constants used in the secure channel protocol, along with a brief description and the default for each constant.

Table 19. Glossary of constants

Constant Name	Description	Value
MSG_COUNTER_WINDOW_SIZE	Maximum number of previously processed mes! sage counters to accept from a given Node and key.	32
MSG_COUNTER_SYNC_REQ_JIT! TER	Maximum amount of random delay before send! ing a MsgCounterSyncReq when the synchroniza! tion request is triggered by receipt of a multicast message.	500 millisec! onds
MSG_COUNTER_SYNC_TIMEOUT	The maximum amount of time (in milliseconds) which a Node SHALL wait for a MsgCounterSyn! cRsp after sending a MsgCounterSyncReq.	400 millisec! onds

4.11. Message Reliability Protocol (MRP)

The Message Reliability Protocol (MRP) provides confirmation of delivery for messages that require reliability. The protocol is optimized for constrained devices that may not be able to receive a mes!

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

sage at the point it is due to be delivered to them. Reliable messaging MAY be enabled on an indi! vidual message basis as required by the protocol design of the higher layer application. Reliability is achieved through time-bounded delivery confirmation, ensuring best effort delivery of critical messages over what may be an inherently lossy and unreliable communication medium.

Flow control mechanisms are not incorporated in MRP because it is intended to be used for short interactions with small numbers of messages in them.

4.11.1. Reliable Messaging Header Fields

The following fields are defined in the Exchange Flags for use exclusively by MRP:

¥ R Flag

Indicates a reliable message. This flag SHALL be set by the sender when a message being sent requires the receiver to send back an acknowledgment. To support unreliable messages, this flag bit MAY be clear, so that no acknowledgements are requested from the receiver.

¥ A Flag

Indicates the message is acting as an acknowledgement. This flag MAY be set on any message. When set, the Acknowledged Message Counter field SHALL be present and valid. This flag SHALL always be set for MRP Standalone Acknowledgement messages.

¥ Acknowledged Message Counter

This field SHALL be set to the Message Counter of the message that is being acknowledged.

4.11.2. Reliable transfer

When the reliability bit is set, the reliable message is transmitted at most MRP_MAX_TRANSMIS! SIONS times until an acknowledgement of receipt is received from the peer or a timeout.

4.11.2.1. Retransmissions

Senders provide an automatic retransmission mechanism for reliable messages. In order for the receiver to receive a message reliably, the sender SHALL trigger the automatic retry mechanism after a period of mrpBackoffTime milliseconds without receiving an acknowledgement, where mrp! BackoffTime is calculated according to the formula below. The sender SHALL retry up to a config! ured maximum number of times (MRP_MAX_TRANSMISSIONS - 1) before giving up and notifying the application.

Messages sent to a Node can be lost for various reasons such as lossy network or insufficient buffer space at the receiver. In the case of sleepy end devices , which wake up infrequently to receive mes! sages destined for them, a sender must be aware of the characteristics of the recipient to ensure it does not attempt to send at a rate beyond the recipientÕs capability. Therefore, the sender SHALL choose retransmission timeouts based on the sleepy characteristics of the destination Node using Section 4.3.2, ÒOperational DiscoveryÓ .

At each sender, a retransmission timer is started each time a reliable message is transmitted. The

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

duration of the retransmission timer SHALL be calculated as follows:

"mrpBackoffTime" = i * "MRP_BACKOFF_BASE"^(max(0,n-"MRP_BACKOFF_THRESHOLD")) * (1.0 + "random"(0,1) * "MRP_BACKOFF_JITTER")

Where:

{:("mrpBackoffTime", =, "the resultant retransmission timeout for this transmission"),(n, =, "the number of send attempts before the current one for this message (0 if this is the initial transmission)"),(i, =, "the base retry interval for the Exchange (either IDLE or ACTIVE"):}

For each unique Exchange, the sender SHALL wait for the acknowledgement message until the retransmission timer, mrpBackoffTime , expires. A sleepy sender SHOULD increase t to also account for its own sleepy interval required to receive the acknowledgment.

The base interval, i , SHALL be set according to the active state of the peer node as stored in the Ses! sion Context of the session (either the Secure Session Context or the Unsecured Session Context depending on the Session Type). The backoff base interval SHALL be set to a value at least 10% greater than the sleep interval of the destination:

- If PeerActiveMode in the Session Context is true:
  - i = SLEEPY_ACTIVE_INTERVAL of the peer
- Else the peer is in idle mode:
  - i = SLEEPY_IDLE_INTERVAL of the peer
- i = MRP_BACKOFF_MARGIN * i

The MRP_BACKOFF_THRESHOLDparameter creates a two-phase scheme which begins with linear backoff to improve initial latency when congestion is not the cause of packet drops, and then transi! tions to exponential backoff to provide convergence when the network is congested. If a positive acknowledgment is received before the retransmission timer expires, the retransmission timer is stopped. Otherwise, if the retransmission timer expires, the message is retransmitted and the timer started again.

The following table illustrates minimum, maximum, and cumulative retransmission times using default parameters.

Table 20. Example MRP Retransmission Times

Metric	Transmission Time [ms]				
Min Jitter	330	330	528	845	1352
Max Jitter	413	413	660	1056	1690
Min Total	330	660	1188	2033	3385
Max Total	413	825	1485	2541	4231

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Metric	Transmission Time [ms]				
Transmission #	0	1	2	3	4

The sender SHOULD initiate Section 4.3.2, ÒOperational DiscoveryÓ in parallel with the first retry to re-resolve the address of the destination Node if the initial transmission fails after one expected round trip. The sender SHOULD use the latest MRP parameters for the destination that result from subsequent Operational Discovery.

4.11.2.2. Acknowledgements

A receiver SHALL acknowledge a reliable message by either using a "piggybacked" acknowledg! ment in the next message destined to the peer, or a standalone acknowledgment, or both.

The acknowledgement message SHALL set the Acknowledged Message Counter field to the value of the Message Counter of the reliable message to be acknowledged.

Piggybacking Acknowledgments on Responses

Acknowledgements MAY be conveyed at the same time (i.e. piggybacked) as data in a response mes! sage. The receiver tries to optimize message transmission by deferring acknowledgments when a reliable message is received (see Section 4.11.5.2.2, ÒStandalone acknowledgement processingÓ) and piggybacking outstanding acknowledgments on messages that it needs to send back (see Section 4.11.5.1.1, ÒPiggyback acknowledgment processingÓ for more details).

Duplicate Message Detection

Since the reliable messaging protocol has a provision for the sender to retransmit messages, there is a significant chance that a duplicate message may arrive at the receiver. The receiver SHALL detect and mark duplicate messages that it receives using the standard authentication and replay protection mechanisms of the secure message layer (see Section 4.5.4, ÒReplay Prevention and Duplicate Message DetectionÓ). The receiver SHALL send an acknowledgment message to the sender for each instance of an authenticated, reliable message, including duplicates. The reliability layer SHALL only propagate the first instance of a message to the next higher layer. Any message marked as a duplicate SHALL be dropped by the reliability layer.

4.11.3. Peer Exchange Management

The Reliable Messaging Protocol operates within the scope of an Exchange between two Nodes. MRP SHALL support one pending acknowledgement and one pending retransmission per Exchange.

MRP control parameters, detailed in Table 21, ÒGlossary of parametersÓ, are computed outside of the Exchange communication itself; instead, they are valid for the duration of a secure session. The SLEEPY_ACTIVE_INTERVAL and SLEEPY_IDLE_INTERVAL, used in computation of MRP control parameters, are determined during Operational Discovery or Section 4.3.1, ÒCommissionable Node DiscoveryÓ. Additionally, the initiator of a secure session MAY provide these parameters in the initial CASE Sig! ma1 or PASE PBKDFParamRequest messages, and the responder MAY provide its parameters in the corresponding protocol messages <ref_Sigma2, CASE Sigma2>> or PBKDFParamResponse.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

4.11.4. Transport Considerations

When the upper layer requests a reliable message over a UDP transport, the R Flag SHALL be set on that message indicating that MRP SHALL be used. Reliable messages sent over TCP or BTP SHALL utilize the underlying reliability mechanisms of those transports and SHOULD NOT set the R Flag.

4.11.5. Reliable Message Processing

4.11.5.1. Reliable Message Processing of Outgoing Messages

To prepare a given Protocol Message for transmission, the message SHALL be processed as follows:

1. Proceed to Section 4.11.5.1.1, ÒPiggyback acknowledgment processingÓ .

Piggyback acknowledgment processing

1. Determine if there is a matching pending acknowledgement in the acknowledgement table for the given message by checking all of the following conditions:
   a. If the Destination Node Id and Exchange Id of the given message and pending acknowledge! ment are the same
   b. AND either
      i. the Session Id and underlying Session Credentials of the given message and pending acknowledgement are the same
      ii. OR both the given message and pending acknowledgement are of Unsecured Session Type.
2. If there is a matching pending acknowledgement, the A Flag SHALL be set on the outbound mes! sage so it will serve as a piggybacked acknowledgement.
   a. For such a piggybacked acknowledgement, the Acknowledgment Message Counter field SHALL be set to the message counter of the received message for which an acknowledge! ment was pending.
   b. If the message being prepared is not a standalone acknowledgement , remove the matching entry from the acknowledgement table .
   c. If the message being prepared is a standalone acknowledgement , set the StandaloneAckSent field of the matching entry in the acknowledgement table to true.

Message retransmission processing

1. If the outbound message is marked to be delivered reliably over a UDP transport, the R Flag SHALL be set on the given message to request an acknowledgement from the peer upon receipt.
   a. Any message flagged for reliable delivery (R Flag set) SHALL be stored in the retransmission table to track the message until it has been successfully acknowledged by the peer.
2. Perform Section 4.6.1, ÒMessage TransmissionÓ processing step on the message to send the mes! sage to the peer:
   a. The same Session ID, Destination Node ID, Security Flags, and transport as were used for the initial message transmission SHALL be used.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

3. If the transport interface returns an error on the send attempt, the error is assessed to deter! mine whether the message can be retried.
   a. If the error is fatal, the application is notified and the message removed from the retrans! mission table .
   b. If there is no error, or a non-fatal error such as no memory, the message is resent
      i. Update the retransmission table to reflect the send count.
      ii. Start a retransmission timer to track the maximum time to wait before attempting another retransmission.
      iii. For each retry, the retransmission table is updated to track the number of retries until the maximum number is attempted, at which point the message is evicted from the retransmission table .

Send flow state diagram

The MRP send flow described above is depicted in the control flow diagram Figure 12, ÒMRP send flowÓ.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

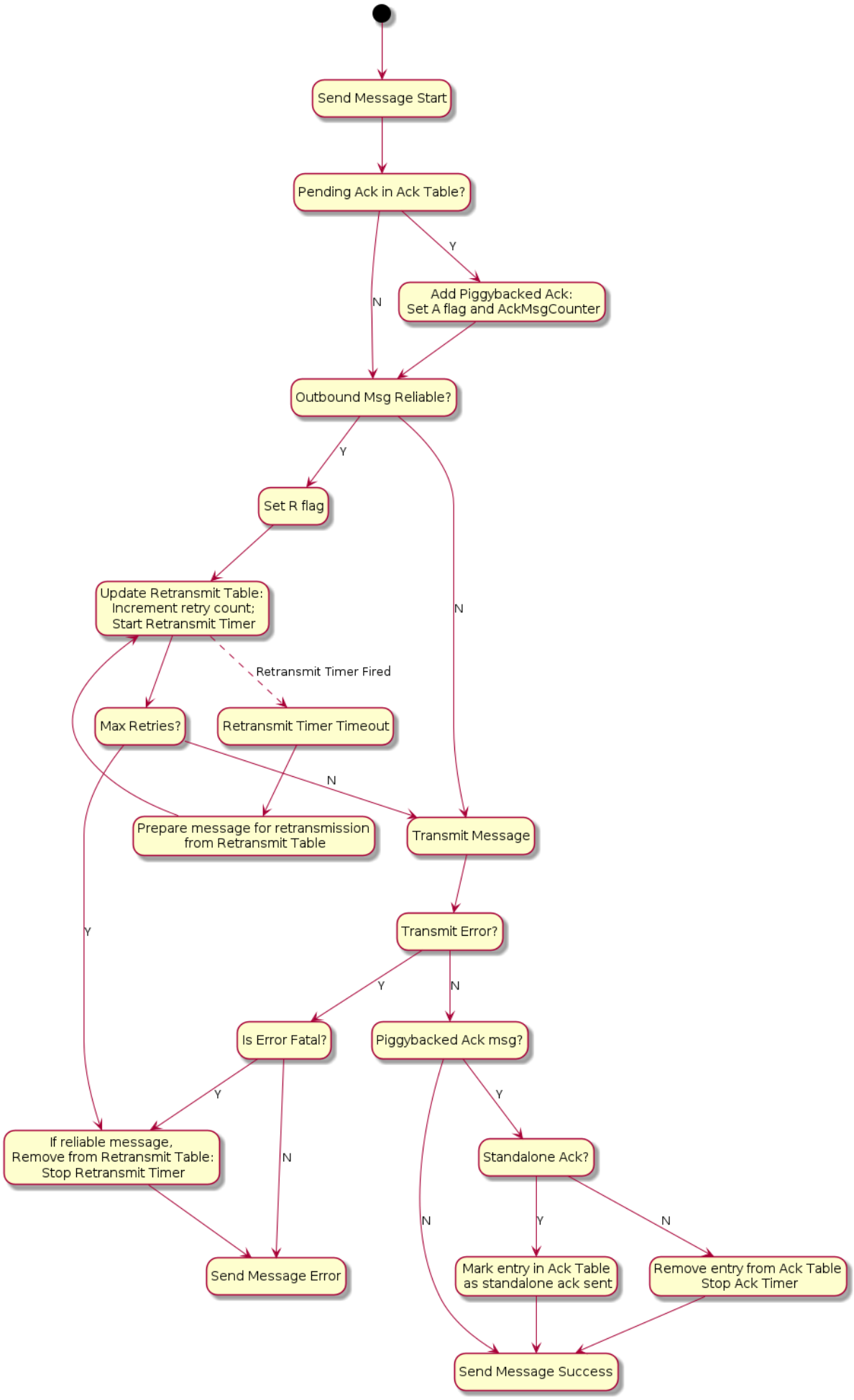


Figure 12. MRP send flow

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

4.11.5.2. Reliable Message Processing of Incoming Messages

A message received from Section 4.6.2, ÒMessage ReceptionÓ for reliability processing SHALL be processed as follows:

1. Verify the message has a legal combination of reliability flags:
   a. If the R Flag is set:
      i. If Group Session Type AND C Flag = 0, drop the message.
   b. If the A Flag is set:
      i. If Group Session Type AND C Flag = 0, drop the message.
2. Proceed to Section 4.9.5.1, ÒExchange Message MatchingÓ.
3. Proceed to Section 4.11.5.2.1, ÒReceived acknowledgement processingÓ .

Received acknowledgement processing

1. If the A Flag is set:
   a. Query the retransmission table for the Acknowledgement Message Counter contained in the received message.
      i. If there is a match:
         A. Remove the entry from the retransmission table .
         B. Stop the retransmission timer for that entry.
      ii. If there is no match, it indicates that this is either a duplicate acknowledgment or the Exchange context does not exist.
2. Proceed to Section 4.11.5.2.2, ÒStandalone acknowledgement processingÓ .

Standalone acknowledgement processing

1. If the R Flag is set, the received message is requesting an acknowledgement be sent back:
   a. If the message is marked as a duplicate:
      i. Immediately send a standalone acknowledgment .
      ii. If the Exchange is marked as an ephemeral exchange the Exchange SHALL be closed.
      iii. Drop the message.
   b. Otherwise, instead of sending an acknowledgement immediately upon the receipt of a reli! able message from a peer, the receiver SHOULD wait for a time no longer than MRP_STAND! ALONE_ACK_TIMEOUT before sending a standalone acknowledgment :
      i. Add the message counter of the received message to the acknowledgement table to signal that an outbound acknowledgement is pending. There can be only one outstanding acknowledgement at a time on a single Exchange. If a pending acknowledgement already exists for the Exchange, and it has StandaloneAckSent set to false, a standalone acknowledgment SHALL be sent immediately for that pending message counter, and the acknowledgement table entry SHALL be replaced for the new message.
      ii. Start the acknowledgement timer for the Exchange.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

A. If the timer triggers before being cancelled, a standalone acknowledgment SHALL be sent to the source of the message. Sending this standalone acknowledgment SHALL NOT remove the acknowledgement table entry and SHALL set the StandaloneAckSent field of the entry to true.

2. The received message is then delivered to the next processing step of Section 4.6.2, ÒMessage ReceptionÓ.

Receive flow state diagram

The MRP receive flow described above is depicted in Figure Figure 13, ÒMRP receive flowÓ.

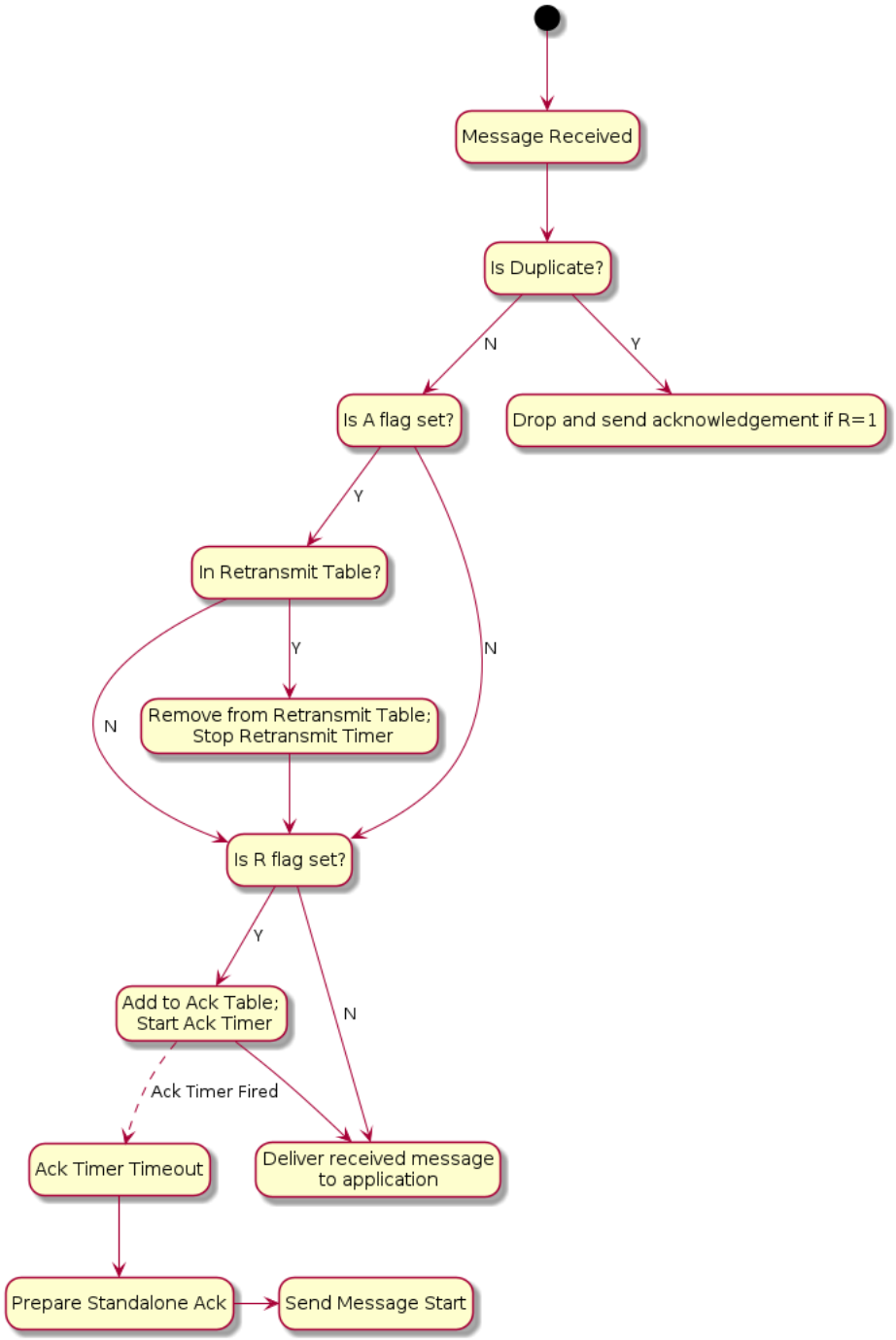


Figure 13. MRP receive flow

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

4.11.6. Reliable Message State

4.11.6.1. Retransmission Table

For retransmissions, the sender maintains a retransmission table of context records containing information on all reliable messages sent that have acknowledgments still pending. Each such reliable message context record includes the following fields:

- Reference to Exchange Context
- Message Counter
- Reference to fully formed, encoded and encrypted message buffer
- Send count
- Retransmission timeout counter

Each time a message that requires acknowledgment is sent, a new retransmission context record is inserted into the retransmission table or an existing record is updated to increment its send count. The message is sent a configurable maximum number of times (MRP_MAX_TRANSMISSIONS) and, if still undelivered, the application is notified of the failure.

4.11.6.2. Acknowledgement Table

The receiver maintains an acknowledgement table of context records containing information on each reliable message for which an acknowledgment SHALL be sent. Each such reliable message context record includes the following fields:

- Reference to Exchange Context
- Message Counter
- A boolean, StandaloneAckSent, indicating whether a standalone acknowledgement has been sent for this message counter. Initially false.

An entry SHALL remain in the table until one of the following things happens:

1. The exchange associated with the entry is closed. See Section 4.9.5.3, ÒClosing an ExchangeÓ.
2. The exchange associated with the entry has switched to track a pending acknowledgement for a new message counter value. See Section 4.11.5.2.2, ÒStandalone acknowledgement processingÓ .
3. A message that is not a standalone acknowledgement is sent which serves as an acknowledgement for the entry. See Section 4.11.5.1.1, ÒPiggyback acknowledgment processingÓ .

4.11.7. MRP Messages

4.11.7.1. MRP Standalone Acknowledgement

The MRP Standalone Acknowledgement message SHALL be formed as follows:

- The application payload SHALL be empty.
- The A Flag SHALL be set to 1.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

- The Acknowledged Message Counter SHALL be included in the header.
- The Protocol ID SHALL be set to PROTOCOL_ID_SECURE_CHANNEL
- The Protocol Opcode SHALL be set to MRP Standalone Acknowledgement .

The rules for when to send this message are detailed in Section 4.11.5.2.2, ÒStandalone acknowledgement processingÓ.

4.11.8. Parameters and Constants

Table 21, ÒGlossary of parametersÓ is a glossary of parameters used in this chapter with a brief description for each parameter. A Node SHALL use the provided default value for each parameter unless the message recipient Node advertises an alternate value for the parameter via Operational Discovery.

Table 21. Glossary of parameters

Parameter Name	Description	Default Value
MRP_MAX_TRANSMISSIONS	The maximum number of transmission attempts for a given reliable message. The sender MAY choose this value as it sees fit.	5
MRP_BACKOFF_BASE	The base number for the exponential backoff equation.	1.6
MRP_BACKOFF_JITTER	The scaler for random jitter in the backoff equation.	0.25
MRP_BACKOFF_MARGIN	The scaler margin increase to backoff over the peer sleepy interval.	1.1
MRP_BACKOFF_THRESHOLD	The number of retransmissions before transitioning from linear to exponential backoff.	1
MRP_STANDALONE_ACK_TIMEOUT	Amount of time to wait for an opportunity to piggyback an acknowledgement on an outbound message before falling back to sending a standalone acknowledgement .	200 milliseconds

4.12. Unicast Communication

This section specifies the semantics of establishing a unicast session and the lifecycle of a unicast session.

Unicast sessions exist in one of two phases:

1. Session Establishment Phase: A series of well-defined unencrypted messages that aim to establish a shared key.
2. Application Data Phase: A series of ad-hoc encrypted messages exchanging interaction model protocol actions, application data, etc.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

4.12.1. Session Establishment Phase

Session establishment uses either the CASE or PASE protocol.

CASE SHALL be used as a session establishment mechanism for all sessions except:

1. Communication for the purpose of commissioning when NOC has not yet been installed

PASE SHALL only be used for session establishment mechanism during device commissioning. PASE SHALL NOT be used as a session establishment mechanism for any other session. BTP MAY be used as the transport for device commissioning. BTP SHALL NOT be used as a transport for opera! tional purposes.

Unless otherwise specified, the CASE, PASE, User-Directed Commissioning protocol, and Secure Channel Status Report messages SHALL be the only allowed unencrypted messages.

This phase aims to:

1. Authenticate peers (CASE-based sessions only).
2. Derive shared secrets to encrypt subsequent session data.
3. Choose session identifiers to identify the subsequent session.

4.12.1.1. Unsecured Session Context

The following session context data SHALL be utilized to associate messages to a particular peer and recover context during unencrypted sessions:

1. Session Role: Records whether the node is the session initiator or responder.
2. Ephemeral Initiator Node ID : Randomly selected for each session by the initiator from the Oper! ational Node ID range and enclosed by initiator as Source Node ID and responder as Destination Node ID .

   %Initiators SHALL select a new random ephemeral node ID for each unsecured session, and SHALL select an ID that does not conflict with any ephemeral node IDs for any other ongo! ing unsecured sessions opened by the initiator.
3. Message Reception State: Provides tracking for the Unencrypted Message Counter of the remote peer.

Matching and responder creation of Unsecured Session Contexts SHALL be as follows:

1. Given an incoming unencrypted message
   a. Locate any Unsecured Session Context with matching Ephemeral Initiator Node ID
      i. If any is located, the incoming message SHALL be assumed to be associated with this Unsecured Session Context
   b. Else if the message carries a Source Node ID
      i. Create a new Unsecured Session Context
      ii. Set Session Role to responder

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

iii. Record the incoming messageÕs Source Node ID as Ephemeral Initiator Node ID

c. Else discard the message

Initiator creation of Unsecured Session Contexts SHALL be as follows:

1. Given the first outgoing message of an unencrypted exchange
   a. Create a new Unsecured Session Context
   b. Set Session Role to initiator
   c. Randomly select a node ID from the Operational Node ID range that does not collide with any ephemeral node IDs for any other ongoing unsecured sessions opened by the initiator and record this as Ephemeral Initiator Node ID

4.12.1.2. Session Establishment over IP

When establishing a session over IP, the initiator SHALL use TCP when both of the following are true:

1. The initiator supports TCP
2. The responder supports TCP as indicated by the T flag

If one or both nodes do not support TCP, the initiator SHALL use MRP to establish the session.

The choice of transport used during session establishment SHALL be used for the transport of mes! sages of the established session.

4.12.1.3. Shared Secrets

Both CASE and PASE produce two shared keys: I2RKey and R2IKey. These keys will be saved to the sessionÕs context and used to encrypt and decrypt messages during the Session Data Phase.

Nodes that support the CASE session resumption SHALL also save to the sessionÕs context the SharedSecret computed during the CASE protocol execution.

4.12.1.4. Choosing Secure Unicast Session Identifiers

Both CASE and PASE allow each participant the ability to choose a unicast session identifier for the subsequent encrypted session. The session identifier SHALL be used to look up the relevant encryp! tion keys and any other metadata for a particular session.

Messages using a unicast session identifier SHALL set the Session Type field to 0. Each peer SHALL specify a Session Identifier unique in reference to their own active sessions. There SHALL NOT be overlap between the Session ID values allocated for PASE and CASE sessions, as the Session Identi! fier space is shared across both session establishment methods.

For example, if the initiator has two active sessions with session identifiers 0x0001 and 0x0002, it could choose any non-zero session identifier besides 0x0001 and 0x0002.

If there are no available session identifiers (i.e. the participant has 65,535 open sessions), the Node SHALL terminate an existing session to free a session identifier.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

4.12.2. Application Data Phase

When the last CASE or PASE protocol message is sent or received and successfully processed, session establishment has completed.

4.12.2.1. Secure Session Context

During the Application Data Phase, the following conceptual session context data SHALL be utilized to securely process subsequent messages:

1. Session Type: Records whether the session was established using CASE or PASE.
2. Session Role: Records whether the node is the session initiator or responder.
3. Local Session Identifier: Individually selected by each participant in secure unicast communication during session establishment and used as a unique identifier to recover encryption keys, authenticate incoming messages and associate them to existing sessions.

   %On a given Node, this is the identifier that SHALL be used to map from an incoming message's Session ID field to the session context data.
4. Peer Session Identifier: Assigned by the peer during session establishment.

   %On a given Node, this is the identifier that SHALL be used in the Session ID field of every outgoing message associated with the session, so that it can be interpreted as the Local Session Identifier by the remote peer.
5. I2RKey: Encrypts data in messages sent from the initiator of session establishment to the responder.
6. R2IKey: Encrypts data in messages sent from the session establishment responder to the initiator.
7. SharedSecret: Computed during the CASE protocol execution and re-used when CASE session resumption is implemented.
8. Local Message Counter: Secure Session Message Counter for outbound messages.

   %At successful session establishment, the Local Message Counter SHALL be initialized per Section 4.5.1.1, "Message Counter Initialization".
9. Message Reception State: Provides tracking for the Secure Session Message Counter of the remote peer.
10. Local Fabric Index: Records the local Index for the session's Fabric, which MAY be used to look up Fabric metadata related to the Fabric for which this session context applies.

    %This field SHALL contain the "no Fabric" value of 0 when the SessionType is PASE and successful invocation of the AddNOC command has not yet occurred during commissioning.
11. Peer Node ID: Records the authenticated node ID of the remote peer, when available.

    %This field SHALL contain the "Unspecified Node ID" value of 0 when the SessionType is PASE
12. Resumption ID: The ID used when resuming a session between the local and remote peer.
13. SessionTimestamp A timestamp indicating the time at which the last message was sent or received. This timestamp SHALL be initialized with the time the session was created. See Section 4.10.1.1, "Session Establishment - Out of Resources" for more information.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

14. ActiveTimestamp: A timestamp indicating the time at which the last message was received. This timestamp SHALL be initialized with the time the session was created.
15. The following sleepy parameters (see Table 5, ÒGlossary of parametersÓ):
    a. SLEEPY_IDLE_INTERVAL
    b. SLEEPY_ACTIVE_INTERVAL
    c. PeerActiveMode: A boolean that tracks whether the peer node is in Active or Idle mode as defined in Section 2.9, ÒSleepy End Device (SED)Ó. PeerActiveMode is set as follows:

```
PeerActiveMode = (now() - ActiveTimestamp) &lt; "SLEEPY_ACTIVE_THRESHOLD"
```

Note that the Local Fabric Index and Peer Fabric Index reported in the NOC Response MAY differ in value, while still referring to the same Fabric, since for a given complete Fabric Reference, the short Fabric Index allocated during commissioning of the respective Nodes on the same Fabric MAY be different. This possible difference is due to the order in which the Fabric in question was joined in the lifecycle of the respective Nodes. See the section on AddNOC command behavior for details on Fabric Index allocation behavior over time.

There SHALL also be reservation of storage to support CASE Authenticated Tag (CAT) fields. The CAT fields are 32-bit values that MAY have been present in RDN case-authenticated-tag of the remote peerÕs operational certificate, during CASE.

The CATfields are used to cache Operational Certificate data so that it can be used by the ACL processing logic to support CASE Authenticated Tags .

Since these fields MAY be omitted from NOCs, they MAY be marked as absent in the context, such that they are not taken into account when missing. When present, they SHALL be stored. Maximum up to 3 CATfields SHALL be supported.

Their value is unused in PASEsession contexts.

4.13. Session Establishment

4.13.1. Passcode-Authenticated Session Establishment (PASE)

This section describes session establishment using a shared passcode together with an augmented Password-Authenticated Key Exchange (PAKE), in which only one party knows the passcode before! hand, to generate shared keys. This protocol is only used when commissioning a Node (i.e. the Com! missionee).

4.13.1.1. Protocol Overview

The Passcode-Authenticated Session Establishment (PASE) protocol aims to establish the first ses! sion between a Commissioner and a Commissionee using a known passcode provided out-of-band. The pairing is performed using Section 3.10, ÒPassword-Authenticated Key Exchange (PAKE)Ó and relies on a Password-Based Key Derivation Function (PBKDF) where the passcode is used as pass! word.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

This session establishment protocol provides a means to:

1. Communicate PBKDF parameters.
2. Derive PAKE bidirectional secrets.

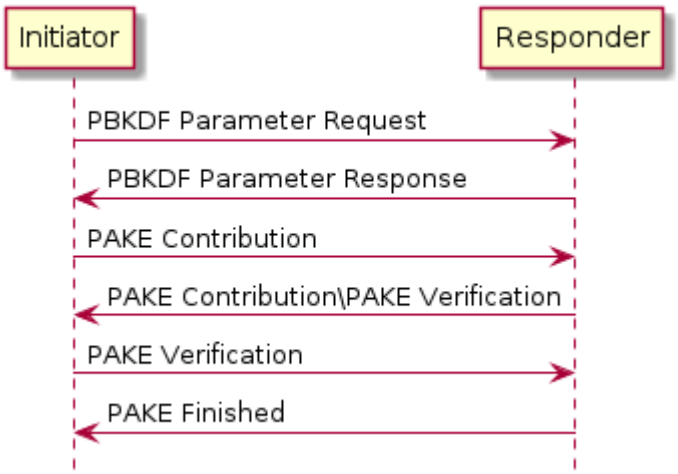


Figure 14. Overview of the PASE Protocol

The Commissioner is the Initiator and the Commissionee is the Responder.

It is assumed that the initiator has somehow obtained the passcode and that the responder has the relevant Crypto_PAKEValues_Responder corresponding to the passcode before starting a PASE session establishment protocol.

4.13.1.2. Protocol Details

Message format

All PASE messages SHALL be structured as specified in Section 4.4, ÒMessage Frame FormatÓ.

All PASE messages are unsecured at the message layer:

- The Session ID field SHALL be set to 0.
- The Session Type bits of the Security Flags SHALL be set to 0.
- The S Flag and DSIZ fields of the Message Flags SHALL be set to 0.

For each PASE message, the application payload is the TLV encoding of the message structure as defined below:

Table 22. PASE Messages

Message Name	Payload TLV Encoding
PBKDFParamRequest	pbkdfparamreq-struct
PBKDFParamResponse	pbkdfparamresp-struct
Pake1	pake-1-struct
Pake2	pake-2-struct
Pake3	pake-3-struct

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Message Name	Payload TLV Encoding
PakeFinished	N/A (encoded via StatusReport)

The other fields of the Message format are not specific to the PASE messages.

For all TLV-encoded PASE messages, any context-specific tags not listed in the associated TLV schemas SHALL be reserved for future use, and SHALL be silently ignored if seen by a recipient which cannot understand them.

Message Exchange

The PBKDFParamRequest, PBKDFParamResponse, Pake1, Pake2, Pake3, and PakeFinished of a distinct session establishment are part of the same message exchange. The initiator and responder SHALL NOT send encrypted application data in the newly established session until PakeFinished is received by the initiator within the unencrypted session used for establishment.

Each message SHALL use PROTOCOL_ID_SECURE_CHANNEL as Protocol ID and the corresponding Protocol Opcode as defined in Table 17, ÒSecure Channel Protocol OpcodesÓ.

The flags of the Exchange Flags of the Protocol Header are defined as follows per PASE message:

Message	I Flag
PBKDFParamRequest	1
PBKDFParamResponse	0
Pake1	1
Pake2	0
Pake3	1

All PASE messages SHALL be sent reliably. This may be implicit (e.g. TCP) or explicit (e.g. MRP reli! able messaging) in the underlying transport.

The other fields of the Protocol Header are not specific to the PASE messages.

PBKDFParamRequest

This message serves to request the PBKDF parameters, with a payload that follows this TLV schema:

```
pbkdfparamreq-struct => STRUCTURE [ tag-order ]
{
Ê initiatorRandom    [1] : OCTET STRING [ length 32 ],
Ê initiatorSessionId [2] : UNSIGNED INTEGER [ range 16-bits ],
Ê passcodeId         [3] : UNSIGNED INTEGER [ length 16-bits ],
Ê hasPBKDFParameters [4] : BOOLEAN,
Ê initiatorSEDParams [5, optional] :          sed-parameter-struct
}
```

1. The initiator SHALL generate a random number InitiatorRandom = Crypto_DRBG(len = 32 * 8) .

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

2. The initiator SHALL generate a session identifier (InitiatorSessionId) for subsequent identification of this session. The InitiatorSessionId field SHALL NOT overlap with any other existing PASE or CASE session identifier in use by the responder. See Section 4.12.1.4, "Choosing Secure Unicast Session Identifiers" for more details. The initiator SHALL set the Local Session Identifier in the Session Context to the value InitiatorSessionId.
3. The initiator SHALL choose a passcode identifier (PasscodeId) corresponding to a particular PAKE passcode verifier installed on the responder. A value of 0 for the passcodeID SHALL correspond to the PAKE passcode verifier for the currently-open commissioning window, if any. Non-zero values are reserved for future use. For example, for initial commissioning, the verifier would be the built-in verifier matching the Onboarding Payload 's passcode or, equivalently, the multi-fabric Basic Commissioning Method passcode if that method is supported. For the multi-fabric Enhanced Commissioning Method, the verifier would match the verifier provided through the OpenCommissioningWindow command.
4. The initiator SHALL indicate whether the PBKDF parameters (salt and iterations) are known for the particular passcodeId (for example from the QR code) by setting HasPBKDFParameters. If HasPBKDFParameters is set to True, the responder SHALL NOT return the PBKDF parameters. If HasPBKDFParameters is set to False, the responder SHALL return the PBKDF parameters.
5. The initiator SHALL send a message with the appropriate Protocol Id and Protocol Opcode from Table 17, "Secure Channel Protocol Opcodes" whose payload is the TLV-encoded pbkdf-paramreq-struct PBKDFParamRequest with an anonymous tag for the outermost struct.

```
PBKDFParamRequest =
{
Ê initiatorRandom (1) = InitiatorRandom,
Ê initiatorSessionId (2) = InitiatorSessionId,
Ê passcodeID (3) = PasscodeId,
Ê hasPBKDFParameters (4) = HasPBKDFParameters,
}
```

PBKDFParamResponse

```
pbkdfparamresp-struct => STRUCTURE [ tag-order ]
{
Ê initiatorRandom    [1] : OCTET STRING [ length 32 ],
Ê responderRandom    [2] : OCTET STRING [ length 32 ],
Ê responderSessionId [3] : UNSIGNED INTEGER [ range 16-bits ],
Ê pbkdf_parameters   [4] : Crypto_PBKDFParameterSet,
Ê responderSEDParams [5, optional] :  sed-parameter-struct
}
```

On receipt of PBKDFParamRequest, the responder SHALL:

1. Verify passcodeID is set to 0. If verification fails, the responder SHALL send a status report : StatusReport(GeneralCode: FAILURE, ProtocolId: SECURE_CHANNEL, ProtocolCode: INVALID_PARAMETER) and perform no further processing.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

2. Generate a random number ResponderRandom = Crypto_DRBG(len = 32 * 8)
3. Generate a session identifier (ResponderSessionId) for subsequent identification of this session. The ResponderSessionId field SHALL NOT overlap with any other existing PASE or CASE session identifier in use by the responder. See Section 4.12.1.4, ÒChoosing Secure Unicast Session Identifiers Ó for more details. The responder SHALL set the Local Session Identifier in the Session Context to the value ResponderSessionId.
4. Set the Peer Session Identifier in the Session Context to the value PBKDFParamRequest.initiatorSessionId .
5. Construct the appropriate Crypto_PBKDFParameterSet(PBKDFParameters). If PBKDFParamRequest.hasPBKDFParameters is True the responder SHALL NOT include the PBKDF parameters (i.e. salt and iteration count) in the Crypto_PBKDFParameterSet. If Msg1.hasPBKDFParameters is False the responder SHALL include the PBKDF parameters (i.e. salt and iteration count) in the Crypto_PBKDFParameterSet
6. Send a message with the appropriate Protocol Id and Protocol Opcode from Table 17, ÒSecure Channel Protocol Opcodes Ó whose payload is the TLV-encoded pbkdfparamresp-struct PBKDFParamResponse with an anonymous tag for the outermost struct.

```
PBKDFParamResponse =
{
Ê initiatorRandom (1) = PBKDFParamRequest.initiatorRandom,
Ê responderRandom (2) = ResponderRandom,
Ê responderSessionId (3) = ResponderSessionId,
Ê pbkdf_parameters (4) = PBKDFParameters
}
```

Pake1

```
pake-1-struct => STRUCTURE [ tag-order ]
{
Ê pA [1] : OCTET STRING [ length CRYPTO_PUBLIC_KEY_SIZE_BYTES ],
}
```

On receipt of PBKDFParamResponse, the initiator SHALL:

1. Set the Peer Session Identifier in the Session Context to the value PBKDFParamResponse.responderSessionId .
2. Generate the Crypto_PAKEValues_Initiator according to the PBKDFParamResponse.pbkdf_parameters
3. Using Crypto_PAKEValues_Initiator , generate pA := Crypto_pA(Crypto_PAKEValues_Initiator)
4. Send a message with the appropriate Protocol Id and Protocol Opcode from Table 17, ÒSecure Channel Protocol Opcodes Ó whose payload is the TLV-encoded pake-1-struct Pake1 with an anonymous tag for the outermost struct.

```
Pake1 =
{
```

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

```
Ê pA (1) = pA,
}
```

Pake2

```
pake-2-struct => STRUCTURE [ tag-order ]
{
Ê pB [1] : OCTET STRING [ length CRYPTO_PUBLIC_KEY_SIZE_BYTES ],
Ê cB [2] : OCTET STRING [ length CRYPTO_HASH_LEN_BYTES],
}
```

On receipt of Pake1, the responder SHALL:

1. Compute pB := Crypto_pB(Crypto_PAKEValues_Responder) using the passcode verifier indicated in PBKDFParamRequest
2. Compute TT := Crypto_Transcript(PBKDFParamRequest, PBKDFParamResponse, Pake1.pA, pB)
3. Compute (cA, cB, Ke) := Crypto_P2(TT, Pake1.pA, pB)
4. Send a message with the appropriate Protocol Id and Protocol Opcode from Table 17, ÒSecure Channel Protocol OpcodesÓ whose payload is the TLV-encoded pake-2-struct Pake2 with an anonymous tag for the outermost struct.

```
Pake2 =
{
Ê pB (1) = pB,
Ê cB (2) = cB,
}
```

Pake3

```
pake-3-struct => STRUCTURE [ tag-order ]
{
Ê cA [1] : OCTET STRING [length CRYPTO_HASH_LEN_BYTES],
}
```

On receipt of Pake2, the initiator SHALL:

1. Compute TT := Crypto_Transcript(PBKDFParamRequest, PBKDFParamResponse, Pake1.pA, Pake2.pB)
2. Compute (cA, cB, Ke) := Crypto_P2(TT, Pake1.pA, Pake2.pB)
3. Verify Pake2.cB against cB. If verification fails, the initiator SHALL send a status report : StatusReport(GeneralCode: FAILURE, ProtocolId: SECURE_CHANNEL, ProtocolCode: INVALID_PARAMETER) and perform no further processing.
4. Send a message with the appropriate Protocol Id and Protocol Opcode from Table 17, ÒSecure Channel Protocol OpcodesÓ whose payload is the TLV-encoded pake-3-struct Pake3 with an anonymous tag for the outermost struct.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

```
Pake3 =
{
Ê cA (1) = cA,
}
```

5. The initiator SHALL NOT send any encrypted application data until it receives PakeFinished from the responder.

On reception of Pake3, the responder SHALL:

1. Verify Pake3.cA against cA. If verification fails, the responder SHALL send a status report : StatusReport(GeneralCode: FAILURE, ProtocolId: SECURE_CHANNEL, ProtocolCode: INVALID_PARAMETER) and perform no further processing.
2. The responder SHALL set SessionTimestamp to a timestamp from a clock which would allow for the eventual determination of the last session use relative to other sessions.
3. The responder SHALL encode and send PakeFinished .

PakeFinished

To indicate the successful completion of the protocol, the responder SHALL send a status report : StatusReport(GeneralCode: SUCCESS, ProtocolId: SECURE_CHANNEL, ProtocolCode: SESSION_ESTABLISHMENT_SUCCESS).

The initiator SHALL set SessionTimestamp to a timestamp from a clock which would allow for the eventual determination of the last session use relative to other sessions.

Session Encryption Keys

After verification of Pake3, each party can compute their sending and receiving session keys as described below:

```
byte SEKeys_Info[] = /* "SessionKeys" */
Ê    {0x53, 0x65, 0x73, 0x73, 0x69, 0x6f, 0x6e, 0x4b,
Ê     0x65, 0x79, 0x73}


I2RKey || R2IKey || AttestationChallenge =
Ê  Crypto_KDF
Ê  (
Ê   inputKey = Ke,
Ê   salt = [],
Ê   info = SEKeys_Info,
Ê   len = 3 * CRYPTO_SYMMETRIC_KEY_LENGTH_BITS
Ê  )
```

1. Each key is exactly CRYPTO_SYMMETRIC_KEY_LENGTH_BITS bits.
2. The initiator SHALL use I2RKey to encrypt and integrity protect messages and the `R2IKey' to

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

decrypt and verify messages.

3. The responder SHALL use R2IKey to encrypt and integrity protect messages and the `I2RKey' to decrypt and verify messages.
4. The AttestationChallenge SHALL only be used as a challenge during device attestation. See Section 6.2.3, ÒDevice Attestation ProcedureÓ for more details.

Upon initial installation of the new PASE Session Keys:

1. The Node SHALL initialize its Local Message Counter in the Session Context per Section 4.5.1.1, ÒMessage Counter InitializationÓ .
2. The Node SHALL initialize the Message Reception State in the Session Context ` and set the synchronized max_message_counter of the peer to 0.

where || indicates message concatenation and [] a zero-length array.

4.13.2. Certificate Authenticated Session Establishment (CASE)

This section describes a certificate-authenticated session establishment (CASE) protocol using Node Operational credentials . This session establishment mechanism provides an authenticated key exchange between exactly two peers while maintaining privacy of each peer. A resumption mechanism allows bootstrapping a new session from a previous one, dramatically reducing the computation required as well as reducing the number of messages exchanged.

4.13.2.1. Protocol Overview

This session establishment protocol provides a means to:

1. Mutually authenticate both peer Nodes
2. Generate cryptographic keys to secure subsequent communication within a session
3. Exchange operational parameters for the session, such as Session Identifier and MRP parameters

The cryptographic protocol mirrors the [SIGMA] protocol and uses the Identity Protection Key (IPK) to provide better identity protection. Briefly, the protocol will:

1. Exchange ephemeral elliptic curve public keys (Sigma1.initiatorEphPubKey and Sigma2.responderEphPubKey) to generate a shared secret
2. Exchange certificates to prove identities (Sigma2.encrypted2.responderNOC and Sigma3.encrypted3.initiatorNOC)
3. Prove possession of the NOC private key by signing the ephemeral keys and NOC (sigma-2-tbsdata and sigma-3-tbsdata)

The basic protocol can be achieved within 2 round trips as shown below:

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

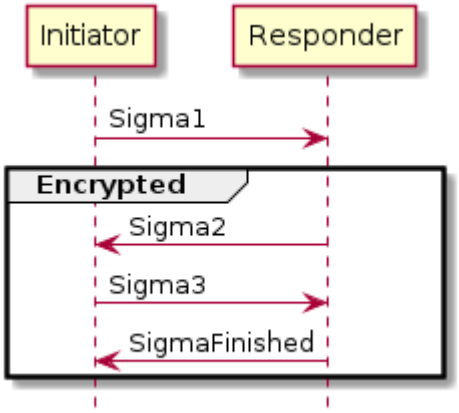


Figure 15. Basic Session Establishment

4.13.2.2. Session Resumption

The protocol also provides a means to quickly resume a session using a previously established session. Resumption does not require expensive signature creation and verification which significantly reduces the computation time. Because of this, resumption is favoured for low-powered devices when applicable. Session resumption SHOULD be used by initiators when the necessary state is known to the initiator.

The nomenclature Sigma1 with Resumption in the following subsections implies a Sigma1 message with both the optional resumptionID and initiatorResumeMIC fields populated in sigma-1-struct .

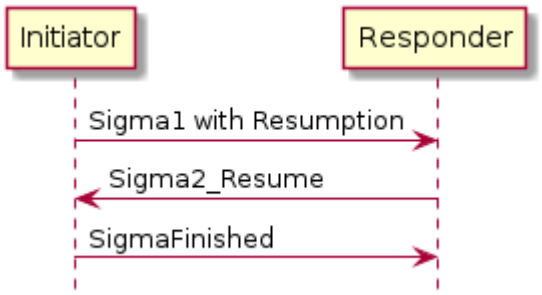


Figure 16. Session Resumption

In the case where a Responder is not able to resume a session as requested by a Sigma1 with Resumption , the information included in the Sigma1 with Resumption message SHALL be processed as a Sigma1 message without any resumption fields to construct a Sigma2 message and continue the standard session establishment protocol without resumption.

To make the resumption succeed, both the Initiator and the Responder SHALL have remembered the SharedSecret they have computed during the previous execution of the CASE session establishment. It SHALL be that SharedSecret that is used to compute the resumption ID.

Session Resumption State

To perform session resumption, the following state from the previous session context must be known to the initiator and responder:

1. SharedSecret
2. Local Fabric Index

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

3. Peer Node ID
4. Peer CASE Authenticated Tags
5. ResumptionID

4.13.2.3. Protocol Details

Message format

All CASE messages SHALL be structured as specified in Section 4.4, ÒMessage Frame FormatÓ.

All CASE messages are unsecured at the message layer:

- ¥ The Session ID field SHALL be set to 0.
- ¥ The Session Type bits of the Security Flags SHALL be set to 0.
- ¥ The S Flag and DSIZ fields of the Message Flags SHALL be set to 0.

For each CASE message, the application payload is the TLV encoding of the message structure as defined below:

Table 23. CASE Messages

Message Name	Payload TLV Encoding
Sigma1	sigma-1-struct
Sigma2	sigma-2-struct ,
Sigma3	sigma-3-struct ,
Sigma2_Resume	sigma-2-resume-struct ,
SigmaFinished	N/A (encoded via StatusReport)

The other fields of the Message format are not specific to the CASE messages.

Message Exchange

The Sigma1, Sigma2, Sigma3, and SigmaFinished of a distinct session establishment are part of the same message exchange. The Sigma1 with resumption , Sigma2_Resume and SigmaFinished of a distinct session resumption are part of the same message exchange. The Sigma1 with resumption , Sigma2, Sigma3 and SigmaFinished of a distinct session resumption that failed to perform the resumption are part of the same message exchange.

Each message SHALL use PROTOCOL_ID_SECURE_CHANNEL as Protocol ID and the corresponding Protocol Opcode as defined in Table 17, ÒSecure Channel Protocol OpcodesÓ.

The Exchange Flags of the Protocol Header are defined as follows per CASE message:

Message	I Flag
CASE Sigma1	1
CASE Sigma2	0

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Message	I Flag
CASE Sigma3	1
CASE Sigma2_Resume	0

For the SigmaFinished message the value of the I Flag is set depending on whether the status mes! sage is sent by the Initiator or the Responder.

All CASE messages SHALL be sent reliably. This may be implicit (e.g. TCP) or explicit (e.g. MRP reli! able messaging) in the underlying transport.

The other fields of the Exchange format are not specific to the CASE messages.

Generate and Send Sigma1

The initiator encodes and sends a Sigma1message, with a payload that follows this TLV schema:

```
sigma-1-struct => STRUCTURE [ tag-order ]
{
Ê initiatorRandom    [1] : OCTET STRING [ length 32 ],
Ê initiatorSessionId [2] : UNSIGNED INTEGER [ range 16-bits ],
Ê destinationId      [3] :          destination-identifier      ,
Ê initiatorEphPubKey [4] :      ec-pub-key,
Ê initiatorSEDParams [5, optional] :         sed-parameter-struct   ,
Ê resumptionID       [6, optional] : OCTET STRING [ length 16 ],
Ê initiatorResumeMIC [7, optional] : OCTET STRING [ length
CRYPTO_AEAD_MIC_LENGTH_BYTES ]
}
```

1. The initiator SHALL generate a random number InitiatorRandom = Crypto_DRBG(len = 32 * 8) .
2. The initiator SHALL generate a session identifier (InitiatorSessionId) for subsequent identifica! tion of this session. The InitiatorSessionId field SHALL NOT overlap with any other existing PASE or CASEsession identifier in use by the initiator. See Section 4.12.1.4, ÒChoosing Secure Unicast Session IdentifiersÓ for more details.
3. The initiator SHALL generate a destination identifier (DestinationId) according to Destination Identifier to enable the responder to properly select a mutual Fabric and trusted root for the secure session.
4. The initiator SHALL generate an ephemeral key pair InitiatorEphKeyPair = Crypto_GenerateKey! pair() .
5. The initiator MAY encode any relevant MRP parameters .
6. Any context-specific tags not listed in the above TLV schemas SHALL be reserved for future use, and SHALL be silently ignored if seen by a responder which cannot understand them.
7. If the initiator is resuming a session from a previous execution of the CASE with the same peer, the initiator SHALL:
   a. Note the ResumptionIDof the previous session.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

b. Generate the S1RK key.

c. Generate the initiatorResumeMIC using the SharedSecret from the previous session:

```
byte Resume1MIC_P[] = {}
byte Resume1MIC_A[] = {}
byte Resume1MIC_Nonce[13] = /* "NCASE_SigmaR1" */
Ê     {0x4e, 0x43, 0x41, 0x53, 0x45, 0x5f, 0x53, 0x69,
Ê      0x67, 0x6d, 0x61, 0x53, 0x31}

InitiatorResume1MIC = Crypto_AEAD_GenerateEncrypt(
Ê    K = S1RK,
Ê    P = Resume1MIC_P,
Ê    A = Resume1MIC_A,
Ê    N = Resume1MIC_Nonce
)
```

8. The initiator SHALL send a message with Secure Channel Protocol ID and Sigma1 Protocol Opcode from Table 17, ÒSecure Channel Protocol OpcodesÓ whose payload is the TLV-encoded Sigma1 Msg1 with an anonymous tag for the outermost struct.

```
Msg1 =
{
Ê initiatorRandom (1) = InitiatorRandom,
Ê initiatorSessionId (2) = InitiatorSessionId,
Ê destinationId (3) = DestinationId,
Ê initiatorEphPubKey (4) = InitiatorEphKeyPair.publicKey
Ê initiatorSEDParams (5) = sed-parameter-struct (optional),
Ê resumptionID (6) = ResumptionID (optional, only present if performing
resumption),
Ê initiatorResumeMIC (7) = InitiatorResume1MIC (optional, only present if
performing resumption)
}
```

Validate Sigma1

On receipt of Msg1, the responder SHALL perform the following:

1. If Msg1 contains either a resumptionID or an initiatorResumeMIC field but not both, the responder SHALL send a status report : StatusReport(GeneralCode: FAILURE, ProtocolId: SECURE_CHANNEL, ProtocolCode: INVALID_PARAMETER) and perform no further processing.
2. Set the Peer Session Identifier in the Session Context to the value Msg1.initiatorSessionId.
3. If Msg1 contains both the resumptionID and initiatorResumeMIC fields, the responder SHALL search for an existing session that has a Resumption ID equal to the incoming resumptionID. If a single such session exists, the responder SHALL follow the steps in Section 4.13.2.3.10, ÒValidate Sigma1 with ResumptionÓ rather than continue the steps outlined in Section 4.13.2.3.5, ÒValidate Sigma1 Destination IDÓ.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

4. If Msg1does not contain a resumptionID and initiatorResumeMIC field, the responder SHALL con! tinue the steps in Section 4.13.2.3.5, ÒValidate Sigma1 Destination IDÓ .

Validate Sigma1 Destination ID

1. The responder SHALL validate the incoming destinationId :
   a. The responder SHALL traverse all its installed Node Operational Certificates (NOC) , gather! ing the associated trusted roots' public keys from the associated chains and SHALL generate a candidateDestinationId based on the procedure in Section 4.13.2.4.1, ÒDestination Identi! fierÓ for that tuple of <Root Public Key, Fabric ID, Node ID>.
   b. The responder SHALL verify that the incoming destinationId matches one of the candidat! eDestinationId generated above. Upon such a match, the associated trusted root, Fabric ID, Node ID and IPK SHALL be recorded for subsequent use.
   c. Note that at the initiator, only the current Epoch Key for the IPK will have been used. At the receiver, several IPK Epoch Keys may be installed, requiring several candidateDestinationId to be computed, one per available IPK Operational Key, per NOC.
2. If there is no candidateDestinationId matching the incoming destinationId , the responder SHALL send a status report : StatusReport(GeneralCode: FAILURE, ProtocolId: SECURE_CHANNEL ProtocolCode: NO_SHARED_TRUST_ROOTS) and perform no further processing.
3. Otherwise, if a match was found for the destinationId , the matched NOC, ICAC (if present), and associated trusted root SHALL be used for selection of the responderNOC and responderICAC in the steps for Sigma2.

Generate and Send Sigma2

If validation is successful, the responder encodes and sends a Sigma2message.

```
sigma-2-tbsdata => STRUCTURE [ tag-order ]
{
Ê responderNOC      [1] : OCTET STRING,
Ê responderICAC     [2, optional] : OCTET STRING,
Ê responderEphPubKey [3] :  ec-pub-key,
Ê initiatorEphPubKey [4] :      ec-pub-key,
}

sigma-2-tbedata => STRUCTURE [ tag-order ]
{
Ê responderNOC  [1] : OCTET STRING,
Ê responderICAC [2, optional] : OCTET STRING,
Ê signature     [3] :        ec-signature  ,
Ê resumptionID  [4] : OCTET STRING [ length 16 ],
}

sigma-2-struct => STRUCTURE [ tag-order ]
{
Ê responderRandom    [1] : OCTET STRING [ length 32 ],
Ê responderSessionId [2] : UNSIGNED INTEGER [ range 16-bits ],
Ê responderEphPubKey [3] :  ec-pub-key,
```


Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

```
Ê encrypted2        [4] : OCTET STRING,
Ê responderSEDParams [5, optional] :    sed-parameter-struct
}
```

NOTE sigma-2-tbsdata is NOT transmitted but is instead signed; the signature will be encrypted and transmitted.

1. The responder SHALL generate a random resumption ID ResumptionID = Crypto_DRBG(len = 16 * 8).
   a. The responder SHALL set the Resumption ID in the Session Context to the value ResumptionID.
2. The responder SHALL use the Node Operational Key Pair ResponderNOKeyPair, responderNOC, and responderICAC (if present) corresponding to the NOC obtained in Section 4.13.2.3.4, ÒValidate Sig!ma1Ó.
3. The responder SHALL generate an ephemeral key pair ResponderEphKeyPair = Crypto_Gener!ateKeypair() .
4. The responder SHALL generate a shared secret :

```
SharedSecret = Crypto_ECDH(
Ê   privateKey = ResponderEphKeyPair.privateKey,
Ê   publicKey = Msg1.initiatorEphPubKey,
)
```

5. The responder SHALL encode the following items as a sigma-2-tbsdata with an anonymous tag :
   a. responderNOC as a matter-certificate
   b. responderICAC (if present) as a matter-certificate
   c. ResponderEphKeyPair.publicKey
   d. Msg1.initiatorEphPubKey
   e. ResumptionID
6. The responder SHALL generate a signature :

```
TBSData2Signature = Crypto_Sign(
Ê   message = sigma-2-tbsdata,
Ê   privateKey = ResponderNOKeyPair.privateKey
)
```

7. The responder SHALL encode the following items as a sigma-2-tbedata , where the encoding of responderNOC and responderICAC items SHALL be byte-for-byte identical to the encoding in sigma-2-tbsdata :
   a. responderNOC as a matter-certificate . This encoding SHALL be byte-for-byte identical to the encoding in sigma-2-tbsdata .
   b. responderICAC (if present) as a matter-certificate . This encoding SHALL be byte-for-byte iden!

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

tical to the encoding in sigma-2-tbsdata .

c. TBSData2Signature

8. The responder SHALL generate the S2Kkey.
9. The responder SHALL generate the encrypted and integrity protected data :

```
byte TBEData2_A[] = {}
byte TBEData2_Nonce[13] = /* "NCASE_Sigma2N" */
Ê     {0x4e, 0x43, 0x41, 0x53, 0x45, 0x5f, 0x53, 0x69,
Ê      0x67, 0x6d, 0x61, 0x32, 0x4e}

TBEData2Encrypted = Crypto_AEAD_GenerateEncrypt(
Ê    K = S2K,
Ê    P = TBEData2,
Ê    A = TBEData2_A,
Ê    N = TBEData2_Nonce
)
```

10. The responder SHALL generate a random number Random = Crypto_DRBG(len = 32 * 8).
11. The responder SHALL generate a session identifier (ResponderSessionId) for subsequent identifi! cation of this secured session. The ResponderSessionIdfield SHALL NOT overlap with any other existing PASE or CASE session identifier in use by the responder. See Section 4.12.1.4, ÒChoosing Secure Unicast Session IdentifiersÓ for more details. The responder SHALL set the Local Session Identifier in the Session Context to the value ResponderSessionId
12. The responder SHALL use the Fabric IPK configured as described in Section 4.13.2.6.1, ÒIdentity Protection Key (IPK)Ó .
13. Any context-specific tags not listed in the above TLV schemas SHALL be reserved for future use, and SHALL be silently ignored if seen by an initiator which cannot understand them.
14. The responder SHALL send a message with Secure Channel Protocol ID and Sigma2Protocol Opcode from Table 17, ÒSecure Channel Protocol OpcodesÓ whose payload is the TLV-encoded Sigma2 Msg2with an anonymous tag for the outermost struct.

```
Msg2 =
{
Ê responderRandom (1) = Random,
Ê responderSessionId (2) = ResponderSessionId,
Ê responderEphPubKey (3) = ResponderEphKeyPair.publicKey,
Ê encrypted2 (4) = TBEData2Encrypted,
Ê responderSEDParams (5) = sed-parameter-struct (optional)
}
```

Validate Sigma2

On receipt of Msg2, the initiator SHALL perform the following:

1. The initiator SHALL generate a shared secret :

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

```
SharedSecret = Crypto_ECDH(
Ê    privateKey = InitiatorEphKeyPair.privateKey,
Ê    publicKey = Msg2.responderEphPubKey,
)
```

2. The initiator SHALL generate the S2K key.
3. The initiator SHALL generate the decrypted data :

```
byte TBEData2_A[] = {}
byte TBEData2_Nonce[13] = /* "NCASE_Sigma2N" */
Ê     {0x4e, 0x43, 0x41, 0x53, 0x45, 0x5f, 0x53, 0x69,
Ê      0x67, 0x6d, 0x61, 0x32, 0x4e}

Success, TBEData2 = Crypto_AEAD_DecryptVerify(
Ê    K = S2K,
Ê    C = Msg2.encrypted2,
Ê    A = TBEData2_A,
Ê    N = TBEData2_Nonce
)
```

4. If the value of Success is FALSE, the initiator SHALL send a status report : StatusReport(GeneralCode: FAILURE, ProtocolId: SECURE_CHANNEL, ProtocolCode: INVALID_PARAMETER) and perform no further processing.
5. The initiator SHALL verify that the NOC in TBEData2.responderNOC and ICAC in TBEData2.responderICAC (if present) fulfills the following constraints:
   a. The Fabric ID and Node ID SHALL match the intended identity of the receiver Node, as included in the computation of the Destination Identifier when generating Sigma1.
   b. If an ICAC is present, and it contains a Fabric ID in its subject, then it SHALL match the FabricID in the NOC leaf certificate.
   c. The certificate chain SHALL chain back to the Trusted Root CA Certificate TrustedRCAC whose public key was used in the computation of the Destination Identifier when generating Sigma1.
   d. All the elements in the certificate chain SHALL respect the Matter Certificate DN Encoding Rules, including range checks for identifiers such as Fabric ID and Node ID.
6. If any of the validations from the previous step fail, the initiator SHALL send a status report : StatusReport(GeneralCode: FAILURE, ProtocolId: SECURE_CHANNEL, ProtocolCode: INVALID_PARAMETER) and perform no further processing.
7. The initiator SHALL verify TBEData2.responderNOC using:
   a. Success = Crypto_VerifyChain(certificates = [TBEData2.responderNOC, TBEData2.responderICAC, TrustedRCAC]), when TBEData2.responderICAC is present, or
   b. Success = Crypto_VerifyChain(certificates = [TBEData2.responderNOC, TrustedRCAC]) , when TBEData2.responderICAC is not present.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

8. If the value of Success is FALSE, the initiator SHALL send a status report : StatusReport(GeneralCode: FAILURE, ProtocolId: SECURE_CHANNEL, ProtocolCode: INVALID_PARAMETER) and perform no further processing.
9. The initiator SHALL encode the following items as a sigma-2-tbsdata with an anonymous tag :
   a. responderNOC as copied from TBEData2
   b. responderICAC (if present) as copied from TBEData2
   c. Msg2.responderEphPubKey
   d. InitiatorEphKeyPair.publicKey
10. The initiator SHALL verify TBEData2.signature (see RFC 5280).

```
Success = Crypto_Verify(
  publicKey = Public key obtained from responderNOC,
  message = sigma-2-tbsdata,
  signature = TBEData2.signature
)
```

11. If the value of Success is FALSE, the initiator SHALL send a status report : StatusReport(GeneralCode: FAILURE, ProtocolId: SECURE_CHANNEL, ProtocolCode: INVALID_PARAMETER) and perform no further processing.
12. Set the Resumption ID in the Session Context to the value TBEData2.resumptionID
13. Set the Peer Session Identifier in the Session Context to the value Msg2.responderSessionId.

Generate and Send Sigma3

If validation is successful, the initiator encodes and sends a Sigma3 message.

```
sigma-3-tbsdata => STRUCTURE [ tag-order ]
{
  initiatorNOC       [1] : OCTET STRING,
  initiatorICAC      [2, optional] : OCTET STRING,
  initiatorEphPubKey [3] :       ec-pub-key,
  responderEphPubKey [4] :   ec-pub-key,
}

sigma-3-tbedata => STRUCTURE [ tag-order ]
{
  initiatorNOC  [1] : OCTET STRING,
  initiatorICAC [2, optional] : OCTET STRING,
  signature     [3] :          ec-signature  ,
}

sigma-3-struct => STRUCTURE [ tag-order ]
{
  encrypted3 [1] : OCTET STRING,
}
```


Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

NOTE sigma-3-tbsdata is NOT transmitted but is instead signed; the signature will be encrypted and transmitted.

1. The initiator SHALL select its Node Operational Key Pair InitiatorNOKeyPair , Node Operational Certificates initiatorNOC and initiatorICAC (if present), and Trusted Root CA Certificate Truste! dRCACorresponding to the chosen Fabric as determined by the Destination Identifier from Sig! ma1.
2. The initiator SHALL encode the following items as a sigma-3-tbsdata with an anonymous tag :
   a. initiatorNOC as a matter-certificate
   b. initiatorICAC (if present) as a matter-certificate
   c. InitiatorEphKeyPair.publicKey
   d. Msg2.responderEphPubKey
3. The initiator SHALL generate a signature :

```
TBSData3Signature = Crypto_Sign(
Ê   message = sigma-3-tbsdata,
Ê   privateKey = InitiatorNOKeyPair.privateKey
)
```

4. The initiator SHALL encode the following items as a sigma-3-tbedata :
   a. initiatorNOC as a matter-certificate . This encoding SHALL be byte-for-byte identical to the encoding in sigma-3-tbsdata .
   b. initiatorICAC (if present) as a matter-certificate . This encoding SHALL be byte-for-byte iden! tical to the encoding in sigma-3-tbsdata .
   c. TBSData3Signature
5. The initiator SHALL generate the S3Kkey.
6. The initiator SHALL generate the encrypted and integrity protected data :

```
byte TBEData3_A[] = {}
byte TBEData3_Nonce[13] = /* "NCASE_Sigma3N" */
Ê     {0x4e, 0x43, 0x41, 0x53, 0x45, 0x5f, 0x53, 0x69,
Ê      0x67, 0x6d, 0x61, 0x33, 0x4e}

TBEData3Encrypted = Crypto_AEAD_GenerateEncrypt(
Ê   K = S3K,
Ê   P = TBEData3,
Ê   A = TBEData3_A,
Ê   N = TBEData3_Nonce
)
```

7. Any context-specific tags not listed in the above TLV schemas SHALL be reserved for future use, and SHALL be silently ignored if seen by a responder which cannot understand them.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

8. The initiator SHALL send a message with Secure Channel Protocol ID and Sigma3 Protocol Opcode from Table 17, ÒSecure Channel Protocol OpcodesÓ whose payload is the TLV-encoded Sigma3 Msg3 = { encrypted3 (1) = TBEData3Encrypted } with an anonymous tag for the outermost struct.
9. The initiator SHALL generate the session encryption keys using the method described in Section 4.13.2.6.6, ÒSession Encryption KeysÓ.

Validate Sigma3

On receipt of Msg3, the responder SHALL perform the following:

1. The responder SHALL generate the S3K key.
2. The responder SHALL generate the decrypted data :

```
byte TBEData3_A[] = {}
byte TBEData3_Nonce[13] = /* "NCASE_Sigma3N" */
Ê     {0x4e, 0x43, 0x41, 0x53, 0x45, 0x5f, 0x53, 0x69,
Ê      0x67, 0x6d, 0x61, 0x33, 0x4e}

Success, TBEData3 = Crypto_AEAD_DecryptVerify(
Ê    K = S3K,
Ê    C = Msg3.encrypted3,
Ê    A = TBEData3_A,
Ê    N = TBEData3_Nonce
)
```

3. If the value of Success is FALSE, the responder SHALL send a status report : StatusReport(GeneralCode: FAILURE, ProtocolId: SECURE_CHANNEL, ProtocolCode: INVALID_PARAMETER) and perform no further processing.
4. The responder SHALL verify that the NOC in TBEData3.responderNOC and the ICAC in TBEData3.responderICAC fulfill the following constraints:
   a. The Fabric ID SHALL match the Fabric ID matched during processing of the Destination Identifier after receiving Sigma1.
   b. If an ICAC is present, and it contains a Fabric ID in its subject, then it SHALL match the FabricID in the NOC leaf certificate.
   c. The certificate chain SHALL chain back to the Trusted Root CA Certificate TrustedRCAC whose public key was matched during processing of the Destination Identifier after receiving Sigma1.
   d. All the elements in the certificate chain SHALL respect the Matter Certificate DN Encoding Rules, including range checks for identifiers such as Fabric ID and Node ID.
5. If any of the validations from the previous step fail, the responder SHALL send a status report : StatusReport(GeneralCode: FAILURE, ProtocolId: SECURE_CHANNEL, ProtocolCode: INVALID_PARAMETER) and perform no further processing.
6. The responder SHALL verify TBEData3.initiatorNOC using:

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

   a. Success = Crypto_VerifyChain(certificates = [TBEData3.initiatorNOC, TBEData3.initiatorICAC, TrustedRCAC]) when TBEData3.initiatorICAC is present, or

   b. Success = Crypto_VerifyChain(certificates = [TBEData3.initiatorNOC, TrustedRCAC]) , when TBEData3.initiatorICAC is not present.

7. If the value of Success is FALSE, the responder SHALL send a status report : StatusReport(GeneralCode: FAILURE, ProtocolId: SECURE_CHANNEL, ProtocolCode: INVALID_PARAMETER) and perform no further processing.
8. The responder SHALL encode the following items as a sigma-3-tbsdata with an anonymous tag
   a. initiatorNOC as copied from TBEData3
   b. initiatorICAC (if present) as copied from TBEData3
   c. Msg1.initiatorEphPubKey
   d. ResponderEphKeyPair.publicKey
9. The responder SHALL verify TBEData3.signature (see RFC 5280):

```
Success = Crypto_Verify(
  publicKey= public key obtained from initiatorNOC,
  message = sigma-3-tbsdata,
  signature = TBEData3.signature
)
```

10. If the value of Success is FALSE, the responder SHALL send a status report : StatusReport(GeneralCode: FAILURE, ProtocolId: SECURE_CHANNEL, ProtocolCode: INVALID_PARAMETER) and perform no further processing.
11. The responder SHALL generate the session keys as described in Section 4.13.2.6.6, ÒSession Encryption KeysÓ .
12. The responder SHALL initialize its Local Message Counter in the Session Context per Section 4.5.1.1, ÒMessage Counter InitializationÓ .
13. The responder SHALL initialize the Message Reception State in the Session Context ` and set the synchronized max_message_counter of the peer to 0.
14. The responder SHALL set SessionTimestamp to a timestamp from a clock which would allow for the eventual determination of the last session use relative to other sessions.
15. The responder SHALL encode and send SigmaFinished .

Validate Sigma1 with Resumption

The responder SHALL continue validating the Sigma1 message Msg1 as follows:

1. Obtain the SharedSecret from the Section 4.12.2.1, ÒSecure Session ContextÓ of the resumed session.
2. Generate the S1RK key .
3. Verify the Resume1MIC by decrypting the following values:

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

```
byte Resume1MIC_A[] = {}
byte Resume1MIC_Nonce[13] = /* "NCASE_SigmaR1" */
Ê     {0x4e, 0x43, 0x41, 0x53, 0x45, 0x5f, 0x53, 0x69,
Ê      0x67, 0x6d, 0x61, 0x53, 0x31}

Success, Resume1MICPayload = Crypto_AEAD_DecryptVerify(
Ê   K = S1RK,
Ê   C = Msg1.sigma1ResumeMIC,
Ê   A = Resume1MIC_A,
Ê   N = Resume1MIC_Nonce
)
```

4. If the value of Success is FALSE, the responder SHALL continue processing Sigma1 as if it didnÕt include any resumption information by continuing the steps in Section 4.13.2.3.5, ÒValidate Sigma1 Destination IDÓ .
5. If the value of Success is TRUE, the responder SHALL:
   a. Set the Peer Session Identifier in the in the Session Context to the value Msg1.initiatorSessionId .
   b. Send a Sigma2_Resume message.

Generate and Send Sigma2_Resume

The responder SHALL encode and send a Sigma2_Resume message in response to a valid Sigma1 with response.

```
sigma-2-resume-struct => STRUCTURE [ tag-order ]
{
Ê resumptionID       [1] : OCTET STRING [ length 16 ],
Ê sigma2ResumeMIC    [2] : OCTET STRING [ length 16 ],
Ê responderSessionID [3] : UNSIGNED INTEGER [ range 16-bits ],
Ê responderSEDParams [4, optional] :     sed-parameter-struct
}
```

1. The responder SHALL generate a new resumption ID ResumptionID = Crypto_DRBG(len = 128).
2. The responder SHALL generate a session identifier (ResponderSessionId) for subsequent identification of this session. The ResponderSessionId field SHALL NOT overlap with any other existing PASE or CASE session identifier in use by the responder. See Section 4.12.1.4, ÒChoosing Secure Unicast Session IdentifiersÓ for more details. The responder SHALL set the Local Session Identifier in the Session Context to the value ResponderSessionId
3. The responder SHALL generate the S2RK key.
4. The responder SHALL generate a resumption MIC :

```
byte Resume2MIC_P[] = {}
byte Resume2MIC_A[] = {}
byte Resume2MIC_Nonce[13] = /* "NCASE_SigmaR2" */
```


Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

```
Ê     {0x4e, 0x43, 0x41, 0x53, 0x45, 0x5f, 0x53, 0x69,
Ê      0x67, 0x6d, 0x61, 0x53, 0x32}

Resume2MIC = Crypto_AEAD_GenerateEncrypt(
Ê    K = S2RK,
Ê    P = Resume2MIC_P,
Ê    A = Resume2MIC_A,
Ê    N = Resume2MIC_Nonce
)
```

5. Any context-specific tags not listed in the above TLV schemas SHALL be reserved for future use, and SHALL be silently ignored if seen by an initiator which cannot understand them.
6. The responder SHALL send a message with the Secure Channel Protocol ID and Sigma2Resume Protocol Opcode from Table 17, ÒSecure Channel Protocol OpcodesÓ whose payload is the TLV-encoded Sigma2_Resume ResumeMsg2 with an anonymous tag for the outermost struct.

```
ResumeMsg2 =
{
Ê resumptionID (1) = ResumptionID,
Ê sigma2ResumeMIC (2) = ResumeMIC2,
Ê responderSessionID (3) = ResponderSessionId,
Ê responderSEDParams (4) = sed-parameter-struct    (optional)
}
```

7. The responder SHALL generate the session keys as described in Section 4.13.2.6.7, ÒResumption Session Encryption KeysÓ .

Validate Sigma2_Resume

On receipt of ResumeMsg2 the initiator SHALL perform the following:

1. The initiator SHALL generate the S2RK key.
2. The initiator SHALL verify the Resume2MIC by decrypting the following values:

```
byte Resume2MIC_A[] = {}
byte Resume2MIC_Nonce[13] = /* "NCASE_SigmaR2" */
Ê     {0x4e, 0x43, 0x41, 0x53, 0x45, 0x5f, 0x53, 0x69,
Ê      0x67, 0x6d, 0x61, 0x53, 0x32}

Success, Resume2MICPayload = Crypto_AEAD_DecryptVerify(
Ê    K = S2RK,
Ê    C = ResumeMsg2.sigma2ResumeMIC,
Ê    A = Resume2MIC_A,
Ê    N = Resume2MIC_Nonce
)
```

3. If Success is FALSE, the initiator SHALL send a status report : StatusReport(GeneralCode: FAILURE,

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

ProtocolId: SECURE_CHANNEL, ProtocolCode: INVALID_PARAMETER) and perform no further processing.

4. The initiator SHALL set the Resumption ID in the Session Context to the value Resume2Msg.resumptionID .
5. The initiator SHALL generate the session keys as described in Section 4.13.2.6.7, ÒResumption Session Encryption KeysÓ .
6. The initiator SHALL reset its Local Message Counter in the Session Context per Section 4.5.1.1, ÒMessage Counter InitializationÓ .
7. The initiator SHALL reset the Message Reception State of the Session Context ` and set the synchronized max_message_counter of the peer to 0.
8. The initiator SHALL set SessionTimestamp to a timestamp from a clock which would allow for the eventual determination of the last session use relative to other sessions.
9. The initiator SHALL set the Peer Session Identifier in the in the Session Context to the value ResumeMsg2.responderSessionId
10. The initiator SHALL send Section 4.13.2.3.13, ÒSigmaFinishedÓ .

SigmaFinished

To indicate the successful completion of the protocol, the Node receiving Sigma3 (if a new session is being established) or Sigma2_Resume (if a session is being resumed) SHALL send a status report : StatusReport(GeneralCode: SUCCESS, ProtocolId: SECURE_CHANNEL, ProtocolCode: SESSION_ESTABLISHMENT_SUCCESS) .

On successful receipt of SigmaFinished :

1. The receiving node SHALL initialize the Local Message Counter according to Section 4.5.1.1, ÒMessage Counter InitializationÓ for the newly established secure session whose success is acknowledged by this message.
2. The receiving node SHALL set SessionTimestamp to a timestamp from a clock which would allow for the eventual determination of the last session usage relative to other sessions.

If this message is received out-of-order or unexpectedly, then it SHALL be ignored.

4.13.2.4. Field Descriptions

Destination Identifier

```
destination-identifier => OCTET STRING [length CRYPTO_HASH_LEN_BYTES]
```

The Destination Identifier field enables the initiator of the Sigma1 message to unambiguously express the following, in a privacy-preserving manner:

- ¥ Which shared Trusted Root to select
- ¥ Which Fabric ID to use for validation of initiator and responder operational certificates
- ¥ Which Node ID is targeted in the given Fabric

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

This serves several purposes:

1. It requires an initiator to have knowledge of both the IPK and one of the full identities of the responder Node before it forces the responder node to generate a costly Sigma2 message
   a. Note that the replay of previously recorded initiator messages is possible, and therefore a Node MAY choose to keep memory of some prior destination identifiers which it would later reject if seen again, for additional replay protection
2. It ensures that there is no ambiguity on the responder as to which Fabric was selected for com! munication
3. It hides which Fabric was chosen by the initiator

A destination identifier is generated by:

1. Concatenating the following octet strings for subsequent usage as a destinationMessage:
   %initiatorRandom : The value of initiatorRandom that will be used in the same message as the Destination Identifier
   %rootPublicKey : The public key of the root of trust of the desired fabric, from the ec-pub-key field of the Matter Certificate of that root, as an uncompressed elliptic curve point as defined in section 2.3.3 of SEC1
   %fabricId : The Fabric ID of the destination, matching the matter-fabric-id field of the Matter Certificate of the desired destinationÕs NOC, and encoding the 64-bit scalar as a little-endian byte order octet string
   %nodeId: The Node ID of the destination, matching the matter-node-id field of the Matter Cer! tificate of the desired destinationÕs NOC, and encoding the 64-bit scalar as a little-endian byte order octet string
2. Obtaining the appropriate Identity Protection Key (IPK) Operational Group Key for the associ! ated Fabric under Group Key Set index 0 within the Group Key Management Cluster .
3. Computing an identifier destinationIdentifier of length CRYPTO_HASH_LEN_BYTES using Crypto_H! MAC()with the IPK as the key and destinationMessage as the message

The above steps can be summarized as:

```
destinationMessage = initiatorRandom || rootPublicKey || fabricId || nodeId
destinationIdentifier = Crypto_HMAC(key=IPK, message=destinationMessage)
```

For example, given the following:

¥ Root public key for the common Fabric, in uncompressed elliptical curve point form:

```
Ê RootPublicKey := // Raw uncompressed point form
Ê   04:4a:9f:42:b1:ca:48:40:d3:72:92:bb:c7:f6:a7:e1:
Ê   1e:22:20:0c:97:6f:c9:00:db:c9:8a:7a:38:3a:64:1c:
Ê   b8:25:4a:2e:56:d4:e2:95:a8:47:94:3b:4e:38:97:c4:
Ê   a7:73:e9:30:27:7b:4d:9f:be:de:8a:05:26:86:bf:ac:
```

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

```
Ê fa
```

¥ Common Fabric ID of 0x2906_C908_D115_D362 scalar (octets "62:d3:15:d1:08:c9:06:29" in little-endian)

¥ Desired Destination Node ID of 0xCD55_44AA_7B13_EF14 (octets "14:ef:13:7b:aa:44:55:cd" in lit! tle-endian)

¥ Identity Protection Key Epoch Key value of:

```
Ê IPKEpochKey := 4a:71:cd:d7:b2:a3:ca:90:24:f9:6f:3c:96:a1:9d:ee
```

%Note that this is the octet string of a group Epoch Key as would be provided in the IpkValue field of the AddNOC command in the Node Operational Credentials Cluster , or in one of the EpochKeyfields of the KeySetWrite command in the Group Key Management Cluster .

%The derived Operational Group Key to be used for computation of a destination identifier, given the above values of root public key, Fabric ID and Identity Protection Key Epoch Key, would be:

```
Ê IPK := 9b:c6:1c:d9:c6:2a:2d:f6:d6:4d:fc:aa:9d:c4:72:d4
```

¥ Initiator Random value of:

```
Ê 7e:17:12:31:56:8d:fa:17:20:6b:3a:cc:f8:fa:ec:2f:
Ê 4d:21:b5:80:11:31:96:f4:7c:7c:4d:eb:81:0a:73:dc
```

Then, using the above procedure would yield the following:

¥ DestinationMessage octets:

```
Ê 7e:17:12:31:56:8d:fa:17:20:6b:3a:cc:f8:fa:ec:2f:
Ê 4d:21:b5:80:11:31:96:f4:7c:7c:4d:eb:81:0a:73:dc:
Ê 04:4a:9f:42:b1:ca:48:40:d3:72:92:bb:c7:f6:a7:e1:
Ê 1e:22:20:0c:97:6f:c9:00:db:c9:8a:7a:38:3a:64:1c:
Ê b8:25:4a:2e:56:d4:e2:95:a8:47:94:3b:4e:38:97:c4:
Ê a7:73:e9:30:27:7b:4d:9f:be:de:8a:05:26:86:bf:ac:
Ê fa:62:d3:15:d1:08:c9:06:29:14:ef:13:7b:aa:44:55:
Ê cd
```

¥ DestinationIdentifier octets:

```
Ê dc:35:dd:5f:c9:13:4c:c5:54:45:38:c9:c3:fc:42:97:
Ê c1:ec:33:70:c8:39:13:6a:80:e1:07:96:45:1d:4c:53
```


Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Public Key

```
ec-pub-key => OCTET STRING [ length CRYPTO_PUBLIC_KEY_SIZE_BYTES
```

A public key ec-pub-key is the byte string representation of an uncompressed elliptic curve point as defined in section 2.3.3 of SEC1.

4.13.2.5. Signature

An ec-signature is the encoding of the signature as defined in Section 3.5.3, ÒSignature and verifica! tionÓ.

```
ec-signature => OCTET STRING [ length ( CRYPTO_GROUP_SIZE_BYTES * 2) ]
```

4.13.2.6. Cryptographic Keys

Identity Protection Key (IPK)

The Identity Protection Key (IPK) SHALL be the operational group key under GroupKeySetID of 0 for the fabric associated with the originatorÕs chosen destination.

The IPK SHALL be exclusively used for Certificate Authenticated Session Establishment . The IPK SHALL NOT be used for operational group communication .

For the generation of the Destination Identifier , the originator SHALL use the operational group key with the second newest EpochStartTime, if one exists, otherwise it SHALL use the single operational group key available.

The operational group key index to use to follow the "second newest EpochStartTime" rule is illus! trated below:

Number of keys in Group Key Set	Operational key index	Epoch Key
1	0	EpochKey0
2	0	EpochKey0
3	1	EpochKey1

Sigma2 Key (S2K)

1. A transcript hash SHALL be generated :

```
TranscriptHash = Crypto_Hash(message = Msg1)
```

2. The Sigma2 key SHALL be generated :

```
byte S2K_Info[] =  /* "Sigma2" */
```


Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

```
Ê    {0x53, 0x69, 0x67, 0x6d, 0x61, 0x32}

S2K = Crypto_KDF(
Ê    inputKey = SharedSecret,
Ê    salt = IPK || Responder Random || Responder Ephemeral Public Key ||
TranscriptHash,
Ê    info = S2K_Info,
Ê    len = CRYPTO_SYMMETRIC_KEY_LENGTH_BITS
)
```

where || indicates message concatenation and IPK is generated according to Section 4.13.2.6.1, ÒIdentity Protection Key (IPK)Ó .

Sigma3 Key (S3K)

1. A transcript hash SHALL be generated :

```
TranscriptHash = Crypto_Hash(message = Msg1 || Msg2)
```

2. The Sigma3 key SHALL be generated :

```
byte S3K_Info[] = /* "Sigma3" */
Ê    {0x53, 0x69, 0x67, 0x6d, 0x61, 0x33}

S3K = Crypto_KDF(
Ê    inputKey = SharedSecret,
Ê    salt = IPK || TranscriptHash,
Ê    info = S3K_Info,
Ê    len = CRYPTO_SYMMETRIC_KEY_LENGTH_BITS
)
```

where || indicates message concatenation and IPK is generated according to Section 4.13.2.6.1, ÒIdentity Protection Key (IPK)Ó .

Sigma1 Resumption Key

The Sigma1 resumption key SHALL be generated :

```
byte S1RK_Info[] = /* "Sigma1_Resume" */
Ê    {0x53, 0x69, 0x67, 0x6d, 0x61, 0x31, 0x5f,
Ê     0x52, 0x65, 0x73, 0x75, 0x6d, 0x65}

S3K_Info
S1RK = Crypto_KDF(
Ê    inputKey = SharedSecret,
Ê    salt = Sigma1.initiatorRandom || ResumptionID,
Ê    info = S1RK_Info,
Ê    len = CRYPTO_SYMMETRIC_KEY_LENGTH_BITS
```

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

```
)
```

where || indicates message concatenation and ResumptionIDis the identifier for the previous ses! sion.

Sigma2 Resumption Key

The Sigma2 resumption key SHALL be generated :

```
byte S2RK_Info[] = /* "Sigma2_Resume" */
Ê    {0x53, 0x69, 0x67, 0x6d, 0x61, 0x32, 0x5f,
Ê     0x52, 0x65, 0x73, 0x75, 0x6d, 0x65}

S2RK = Crypto_KDF(
Ê    inputKey = SharedSecret,
Ê    salt = Sigma1.initiatorRandom || ResumptionID,
Ê    info = S2RK_Info,
Ê    len = CRYPTO_SYMMETRIC_KEY_LENGTH_BITS
)
```

where || indicates message concatenation and ResumptionIDis the new identifier for the this ses! sion.

Session Encryption Keys

1. A transcript hash SHALL be generated :

   ```
   TranscriptHash = Crypto_Hash(message = Msg1 || Msg2 || Msg3)
   ```

2. The session encryption keys SHALL be generated :

   ```
   byte SEKeys_Info[] = /* "SessionKeys" */
   Ê    {0x53, 0x65, 0x73, 0x73, 0x69, 0x6f, 0x6e, 0x4b,
   Ê     0x65, 0x79, 0x73}

   I2RKey || R2IKey || AttestationChallenge = Crypto_KDF(
   Ê    inputKey = SharedSecret,
   Ê    salt = IPK || TranscriptHash,
   Ê    info = SEKeys_Info,
   Ê    len = 3 * CRYPTO_SYMMETRIC_KEY_LENGTH_BITS
   )
   ```

3. Each key is exactly CRYPTO_SYMMETRIC_KEY_LENGTH_BITS bits.
4. The initiator SHALL use I2RKeyto encrypt and integrity protect messages and the `R2IKey' to decrypt and verify messages.
5. The responder SHALL use R2IKeyto encrypt and integrity protect messages and the `I2RKey' to

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

decrypt and verify messages.

6. The AttestationChallenge SHALL only be used as a challenge during device attestation. See Section 6.2.3, ÒDevice Attestation ProcedureÓ for more details.

where || indicates message concatenation.

Resumption Session Encryption Keys

1. The resumption session encryption keys SHALL be generated :

```
byte RSEKeys_Info[] = /* "SessionResumptionKeys" */
Ê    {0x53, 0x65, 0x73, 0x73, 0x69, 0x6f, 0x6e, 0x52,
Ê     0x65, 0x73, 0x75, 0x6d, 0x70, 0x74, 0x69, 0x6f,
Ê     0x6e, 0x4b, 0x65, 0x79, 0x73}

I2RKey || R2IKey || AttestationChallenge = Crypto_KDF(
Ê    inputKey = SharedSecret,
Ê    salt = Sigma1.initiatorRandom || ResumptionID,
Ê    info = RSEKeys_Info,
Ê    len = 3 * CRYPTO_SYMMETRIC_KEY_LENGTH_BITS
)
```

2. Each key is exactly CRYPTO_SYMMETRIC_KEY_LENGTH_BITS bits.
3. The initiator SHALL use I2RKey to encrypt and integrity protect messages and the `R2IKey' to decrypt and verify messages.
4. The responder SHALL use R2IKey to encrypt and integrity protect messages and the `I2RKey' to decrypt and verify messages.
5. The AttestationChallenge SHALL only be used as a challenge during device attestation. See Section 6.2.3, ÒDevice Attestation ProcedureÓ for more details.

where || indicates message concatenation and ResumptionID is the new identifier for the this session.

4.13.2.7. Session Context Storage

After the session is established successfully at both peers, some fields SHALL be recorded in the secure session context for later use (see Section 4.12.2, ÒApplication Data PhaseÓ), in addition to the Session Encryption Keys :

- The peer NOC's matter-node-id (1.3.6.1.4.1.37244.1.1) from the subject field
- The Section 2.5.1, ÒFabric References and Fabric IdentifierÓ for the Fabric within which this secure session is being established
- All peer NOC's case-authenticated-tag (1.3.6.1.4.1.37244.1.6) from the subject field, if present

These fields MAY be recorded at any opportune point during the protocol, but SHALL only be committed to the secure session context once the session is established successfully at both peers.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

4.13.2.8. Minimal Number of CASE Sessions

A node SHALL support at least 3 CASE session contexts per fabric. Device type specifications MAY require a larger minimum. Unless the device type specification says otherwise, a minimum number it defines is a per-fabric minimum.

The minimal supported capabilities, subject to the minimal constraints above, are reported in the CapabilityMinima of the Basic Information cluster .

¥ Example: If a device type requires at least 4 CASE session contexts, and a node supports 7 fab! rics, the node would support at least 28 CASE session contexts, and ensure that each fabric could use at least 4 of them.

4.14. Group Communication

This section specifies the semantics of sending and receiving multicast group messages and the life! cycle of such groupcast sessions. Multicast addressing is accomplished using the 16-bit Group ID field as the destination address. A multicast group is a collection of Nodes, all registered under the same multicast Group ID. A multicast message is sent to a particular destination group and is received by all members of that group.

4.14.1. Groupcast Session Context

Groupcast sessions are conceptually long-running, lasting the duration of a nodeÕs membership in a group. Group membership is tracked in the Group Key Management Cluster . However, on ingress of each groupcast message, the following ephemeral context SHALL be constructed to inform upper layers of groupcast message provenance:

1. Fabric Index : Records the local Fabric Index for the Fabric to which an incoming messageÕs group is scoped.
2. Group ID: Captures the Group ID to which a groupcast message was sent.
3. Source Node ID: The Source Node ID enclosed by the sender of a groupcast message.

   %Together, Fabric Index , Group ID and Source Node ID comprise a unique identifier that upper layers may use to understand the source and destination of groupcast messages.
4. Source IP Address : The unicast source IP address for the originator of the message.
5. Source Port : The source port for the originator of the message.

   %The source IP address and port MAY be used for unicast responses to group communication peers, as are required for the Message Counter Synchronization Protocol .
6. Operational Group Key : The Operational Group Key that was used to encrypt the incoming group message.
7. Group Session ID : Records the Group Session ID derived from the Operational Group Key used to encrypt the message.

Once a Groupcast Session Context with trust-first policy is created to track authenticated messages from a given Source Node ID, that record SHALL NOT be deleted or recycled until the node reboots. This is to prevent replay attacks that first exhaust the memory allocated to group session counter

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

tracking and then inject older messages as valid, and enforces that trust-first authentication works as intended within the full duration of a boot cycle. Any message from a source that cannot be tracked SHALL be dropped.

4.14.2. Sending a group message

To prepare a multicast message to a Group ID with a given GroupKeySetID and IPv6 hop count parameter, the Node SHALL:

1. Obtain, for the given GroupKeySetID, the current Operational Group Key as the Encryption Key, and the associated Group Session ID.
   a. If no key is found for the given GroupKeySetID, security processing SHALL fail and no fur! ther security processing SHALL be done on this message.
2. Perform Section 4.6.1, ÒMessage TransmissionÓ processing steps on the message with the follow! ing arguments:
   a. The Destination Node Id argument SHALL be the Group Node Id corresponding to the given Group ID.
   b. The Session Id argument SHALL be the Group Session ID from step 1.
   c. The Security Flags SHALL have only the P Flag set.
   d. The transport SHALL be a site-local routable IPv6 interface.

Next, prepare the message as an IPv6 packet:

1. Set the private, secured message from step 2 above as the IPv6 payload.
2. Set the IPv6 hop count to the value given.
3. Set the IPv6 destination to the Section 2.5.6.2, ÒIPv6 Multicast AddressÓ based on the provided destination Group ID, Fabric ID, and Section 11.2.6.2.9, ÒGroupKeyMulticastPolicyÓ of the group key.
4. Set the IPv6 source to an operational IPv6 Unicast Address of the sending Node.
5. Set the IPv6 UDP port number to the Matter IPv6 multicast port .

4.14.3. Receiving a group message

All Nodes supporting groups SHALL register to receive on the associated IPv6 multicast address , at the Matter IPv6 multicast port , for each group of which they are a member.

Upon receiving an IPv6 message addressed to one of these Multicast Addresses the Node is regis! tered for, the Node SHALL:

1. Extract the message from the IPv6 payload.
2. Perform Section 4.6.2, ÒMessage ReceptionÓ processing steps on the message.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

4.15. Group Key Management

This section describes operational group keys, a mechanism for generating, disseminating and managing shared symmetric keys across a group of Nodes in a Fabric. Operational group keys are made available to applications for the purpose of authenticating peers, securing group communica! tions, and encrypting data. These keys allow Nodes to:

- ¥ Prove to each other that they are members of the associated group
- ¥ Exchange messages confidentially, and without fear of manipulation by non-members of the group
- ¥ Encrypt data in such a way that it can only be decrypted by other members of the group

A central feature of operational group keys is the ability to limit access to keys to a trusted set of Nodes. In particular, credentials required to generate operational group keys SHALL only be acces! sible to Nodes with a certain level of privilege$Ñ$those deemed a member of the group. Barring soft! ware error or compromise of a privileged Node, access to shared keys SHALL be computationally infeasible for non-trusted parties.

Operational group keys are shareable across all types and combinations of Nodes as determined by the Administrator managing the group. Given all Nodes in possession of the current epoch keys for the group can communicate with other Nodes in the group, it is the responsibility of the Adminis! trator managing the group to only compose groups of Nodes where communication is appropriate for the given application and security requirements.

4.15.1. Operational Groups

An operational group is a logical collection of Nodes that are running one or more common applica! tion clusters and share a common security domain in the form of a shared, symmetric group key. For example, a set of Nodes running a lighting application can form an operational group by shar! ing a common operational group key derived from the mechanisms described here. Subgroups can be formed within the operational group by defining distinct Group Identifiers for each set of Nodes, while sharing a common operational group key.

Membership in the security domain of an operational group is determined by a NodeÕs possession of all the epoch keys required to generate the current, valid operational group key for the group. Individual Nodes can be members of multiple operational groups simultaneously. The set of groups to which a Node belongs can change over time as dictated by application requirements and policies. Groups MAY be introduced or withdrawn over time as need arises. === Operational Group Ids Oper! ational groups are identified uniquely within a Fabric by a Group Identifier . Administrators SHALL assign Group Ids such that no two operational groups within a Fabric have the same Group ID. It is assumed a given Administrator has sufficient access to centralized knowledge, so as to allocate unique Group Ids under a given Fabric such that there are no collisions.

Multiple operational groups MAY share the same operational group key, and thus be used to create logical subgroups over a shared security domain. Operational groups which do not share related functionality, such as a lighting group and a sprinkler group, SHOULD NOT share the same opera! tional key. As an example policy, a lighting application could have all lighting Nodes share a single group key, while organizing lighting subgroups for various rooms or spaces within the structure by

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

assigning a different Group ID to each such subgroup.

4.15.2. Operational Group Key Derivation

An operational group key is a symmetric key used as the Encryption Key during Message Processing for group communication. An operational group key is produced by applying a key derivation func! tion with an epoch key and salt value as inputs as follows:

```
OperationalGroupKey =
Ê       Crypto_KDF
Ê       (
Ê          InputKey =          Epoch Key
Ê          Salt =          CompressedFabricIdentifier  ,
Ê          Info = "GroupKey v1.0",
Ê          Length = CRYPTO_SYMMETRIC_KEY_LENGTH_BITS
Ê       )
```

where [] denotes a zero-length array.

The Info portion of the key derivation is specified in Section 4.15.2.1, ÒGroup Security InfoÓ. The Salt portion of the key derivation is specified in Section 4.3.2.2, ÒCompressed Fabric IdentifierÓ .

For example, given:

- ¥ An Epoch Key value of: 23:5b:f7:e6:28:23:d3:58:dc:a4:ba:50:b1:53:5f:4b
- ¥ A CompressedFabricIdentifier value of: 87:e1:b0:04:e2:35:a1:30

After the above derivation following the definition of Crypto_KDFin Section 3.8, ÒKey Derivation Function (KDF)Ó, the resulting operational group key would be: a6:f5:30:6b:af:6d:05:0a:f2:3b:a4:bd:6b:9d:d9:60 .

Group membership is enforced by limiting access to the epoch keys. Only Nodes that possess the input epoch key can derive a given operational key. Lack of possession of a particular epoch key restricts access, based on the distribution policy of the epoch keys.

The following diagram shows the process by which operational keys are derived from the epoch key material:

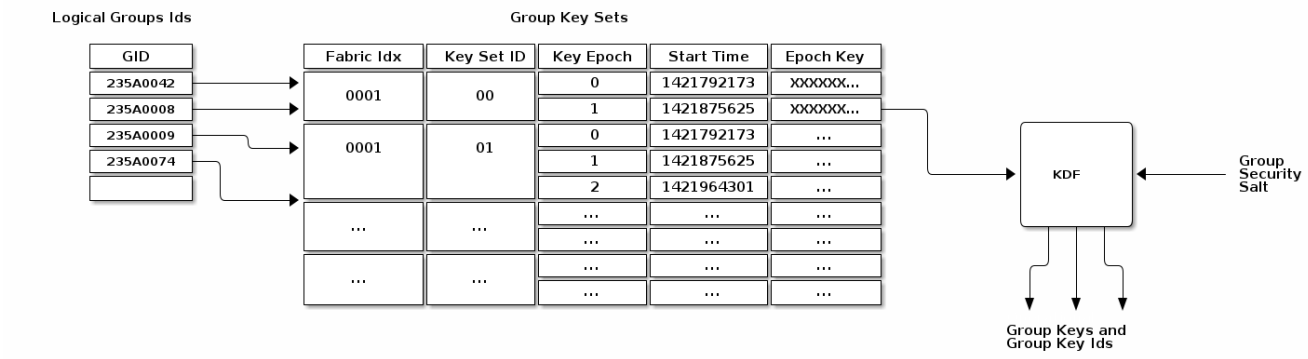


Figure 17. Group Key Derivation

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

4.15.2.1. Group Security Info

A hard-coded group security info is used to diversify the set of operational group keys. This value is hard-coded into the standardÕs implementation, and thus is distributed with the associated code. Should the standardÕs security mechanisms need to evolve (e.g. to upgrade encryption from AES-128 to AES-256), the group security info can be changed to ensure that new keys will be derived for use in the new algorithm. The group security info SHALL be the byte stream "GroupKey v1.0", i.e. 0x47 0x72 0x6f 0x75 0x70 0x4b 0x65 0x79 0x20 0x76 0x31 0x2e 0x30 .

With the exception of the group security info , all input key material SHALL be maintained on a per-Fabric basis.

4.15.2.2. Group Key Set

A group key set limits the key derivation process to Nodes within the respective operational groups. Access to a group key set is limited based on the functionality provided by a Node and/or the privi! lege afforded to it. For example, certain home security devices, such as a security system or door lock, may have access to a "Physical Access" group key set , while devices such as light bulbs or win! dow coverings would not.

Operational group key lifetime is limited by assigning an expiration time to each epoch key in a given group key set . By constraining the validity of a given epoch key to an epoch, the ability for members to derive and operate with an operational group key can be constrained to particular periods of time. Epoch keys may be rotated on a periodic basis, and denying access to updated ver! sions of these keys serves as a means to eject group members.

4.15.3. Epoch Keys

Epoch keys provide a means for limiting the lifetime of derived operational group keys. They also provide a way for an Administrator to revoke access to Nodes that have been explicitly excluded from an operational group (albeit after a period of time).

Epoch keys are generated, managed, and stored by an Administrator on a per-Fabric basis. Each key SHALL be a random value of length CRYPTO_SYMMETRIC_KEY_LENGTH_BITS bits.

```
EpochKey = Crypto_DRBG(len = CRYPTO_SYMMETRIC_KEY_LENGTH_BITS)
```

Each epoch key has associated with it a start time that denotes the time at which the key becomes active for use by transmitting Nodes. Epoch key start times are absolute UTC time in microseconds encoded using the epoch-us representation.

4.15.3.1. Using Epoch Keys

Nodes sending group messages SHALL use operational group keys that are derived from the current epoch key (specifically, the epoch key with the latest start time that is not in the future). Nodes that cannot reliably keep track of time calculate the current epoch key as described in Section 4.15.3.4, ÒEpoch Key Rotation without Time SynchronizationÓ .

Nodes receiving group messages SHALL accept the use of any key derived from one of the currently

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

installed epoch keys. This requirement holds regardless of whether the start time for the key is in the future or the past. This means Nodes continue to accept communication secured under an epoch key until that key is withdrawn by explicitly deleting the key from a NodeÕs group state by the key distribution Administrator.

Note that there is no end time associated with an epoch key. An epoch key marked with the maxi! mum start time SHALL be disabled and render the corresponding epoch key slot unused.

4.15.3.2. Managing Epoch Keys

The epoch keys are managed using the Group Key Management Cluster . For every group key set published by the key distribution Administrator, there SHALL be at least 1 and at most 3 epoch keys in rotation. Key additions or updates are done using the KeySetWrite command.

Key updates are idempotent operations to ensure the Administrator is always the source of truth. An epoch key update SHALL order the keys from oldest to newest.

Any epoch key update MAY deliver a partial key set but SHALL include EpochKey0 and MAY include EpochKey1 and EpochKey2 Any update of the key set, including a partial update, SHALL remove all previous keys in the set, however many were defined.

An Administrator MAY completely remove a group key set from a Node using the KeySetRemove command.

4.15.3.3. Epoch Key Rotation

The key distribution Administrator generates new epoch keys on a regular basis, giving each a unique id and adding them to the list of existing epoch keys within a group. The start time for each new epoch key is scheduled to occur after a configurable key propagation interval . The propagation interval is set sufficiently large such that the Administrator can synchronize all Nodes in the opera! tional group with the new epoch key list within that time.

The rotation rate for epoch keys is expected to be on the order of days to weeks for typical applica! tions, but the rate is configurable as required by the key distribution Administrator. Because of the relatively long rotation interval, and the overlap of active epoch keys, local clock drift within Nodes is generally not a concern.

4.15.3.4. Epoch Key Rotation without Time Synchronization

Although epoch keys are distributed with an associated start time, it is nonetheless possible for Nodes that do not maintain a synchronized clock to participate in key rotation. Specifically, upon receiving a new epoch key list from the key distribution Administrator, such a Node can note which of the keys is the current epoch key by comparing their relative start times and using the current epoch key which has the second newest time. It can then use the current key for all locally initiated security interactions until such time as it makes contact with the distribution Administrator again.

This scheme requires the Node to receive epoch keys from the key distribution Administrator at a rate that is at least as fast as the configured key propagation interval . The Administrator SHOULD provide a sufficient set of epoch keys to Nodes that do not maintain synchronized time so that they can maintain communication with other group members while a key update is in progress. The key

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

distribution Administrator SHOULD update all Nodes without time, such as SEDs, before the new epoch key is activated, and then let Nodes with time all roll to the new epoch key at the synchronized start time.

4.15.3.5. Epoch Key Rotation logic

The full life-cycle of Epoch Key rotation is shown in Figure 18, ÒEpoch Key RotationÓ. For each epoch key slot, the start time of the key is shown as one of the following values:

- ¥ New - a key with a start time in the future.
- ¥ Current - the active key with the newest start time.
- ¥ Previous - the active key with the second newest start time.
- ¥ Old - an active key with the third newest start time.

The diagram shows two types of state transitions:

1. Admin - an update of an old key by the Administrator. Changes made during this state transi! tion are indicated in green.
2. Epoch Activate - activation of an epoch key due to system time becoming greater than the start time. Changes during this state transition are indicated in yellow.

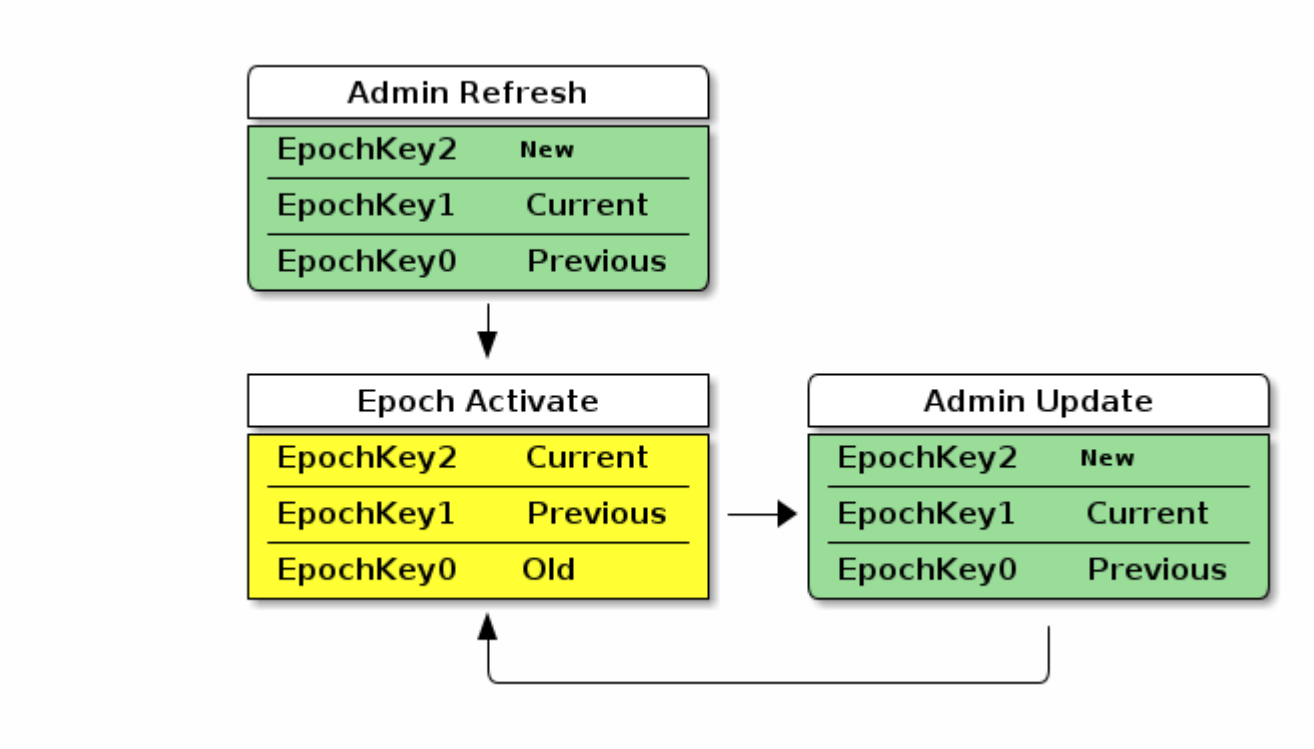


Figure 18. Epoch Key Rotation

The Admin Refresh state begins when an entire group key set is freshly written to a Node during commissioning or administration, such as when a new group is added. The Epoch Activate state is entered when time progresses to activate a fresh current epoch key , aging out the other epoch key slots. The Admin Updatestate is entered when an Administrator updates an old epoch key with a new epoch key. When in steady state, the Admin Refresh state MAY be entered in place of an Admin Update state transition to update additional keys to the required ones or to completely reset the group secu!

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

rity.

Note that in the above diagram, only an example key distribution scheme is illustrated. It is also possible to devise key distribution algorithms that to not rely on time synchronization, or group configurations that never rotate keys in favor or configuring new groups and removing groups and group key sets with expired keys.

Group Key Set ID

The Group Key Set ID is a 16-bit field that uniquely identifies the set of epoch keys used for derivation of an operational group key. Each Group Key Set ID is scoped to a particular Fabric and assigned by an Administrator so as to be unique within a Fabric.

The Group Key Set ID of 0 SHALL be reserved for managing the Identity Protection Key (IPK) on a given Fabric. It SHALL NOT be possible to remove the IPK Key Set if it exists.

4.15.3.6. Group Session ID

A Group Session ID is a special case of a 16-bit Session ID that is specifically used for group communication. When Session Type is 1, denoting a group session, the Session ID SHALL be a Group Session ID as defined here. The Group Session ID identifies probable operational group keys across a Fabric. The Group Session ID for a given operational group key is derived by treating the output of a Crypto_KDF against the associated Operational Group Key as a big-endian representation of a 16-bit integer, as follows:

```
GroupKeyHash =
Ê          Crypto_KDF
Ê          (
Ê              InputKey = OperationalGroupKey,
Ê              Salt = [],
Ê              Info = "GroupKeyHash",
Ê              Length = 16  // Bits
Ê          )

// GroupKeyHash is an array of 2 bytes (16 bits) per Crypto_KDF

// GroupSessionId is computed by considering the GroupKeyHash as a Big-Endian
// value. GroupSessionId is a scalar. Its use in fields within messages may cause a
// re-serialization into a different byte order than the one used for initial
generation.
GroupSessionId = (GroupKeyHash[0] << 8) | (GroupKeyHash[1] << 0)
```

where [] denotes a zero-length array.

For example, given:

¥ An Operational Group Key value of: a6:f5:30:6b:af:6d:05:0a:f2:3b:a4:bd:6b:9d:d9:60

After the above derivation following the definition of Crypto_KDFin Section 3.8, ÒKey Derivation Function (KDF)Ó, the resulting Group Session ID data would be:

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

¥ Raw output of GroupKeyHash: b9:f7

¥ Group Session ID scalar value to be used for further processing: 0xB9F7 (47607 decimal)

The Group Session ID MAY help receiving nodes efficiently locate the Operational Group Key used to encrypt an incoming groupcast message. It SHALL NOT be used as the sole means to locate the asso! ciated Operational Group Key , since it MAY collide within the fabric. Instead, the Group Session ID provides receiving nodes a means to identify Operational Group Key candidates without the need to first attempt to decrypt groupcast messages using all available keys.

On receipt of a message of Group Session Type, all valid, installed, operational group key candidates referenced by the given Group Session ID SHALL be attempted until authentication is passed or there are no more operational group keys to try. This is done because the same Group Session ID might arise from different keys. The chance of a Group Session ID collision is 2^{-16} but the chance of both a Group Session ID collision and the message MIC matching two different operational group keys is 2^{-80}.

Group Session Ids are sized to fit within the context of the Session Identifier field of a message. When used in this context, the Group Session ID value allows a receiving Node to identify the appropriate message encryption key to use from the set of active operational keys it has currently installed.

4.15.4. Distribution of Key Material

The operational group keys mechanism relies on a key distribution Administrator to reliably dis! tribute select epoch key material to appropriate participants. It is assumed the key distribution Administrator is in possession of all epoch keys, has knowledge of the set of group security domain members which require access to those keys, and is responsible for pushing updates of these cre! dentials to all authorized Nodes in those groups it manages.

Key material is distributed to key holders using the Group Key Management Cluster . In general, the key material of a Node is exposed via Attributes with ACL entries that only allow access by the key distribution Administrator. The information exposed in the Section 11.2, ÒGroup Key Management ClusterÓ includes the group epoch keys and associated group session identifiers.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

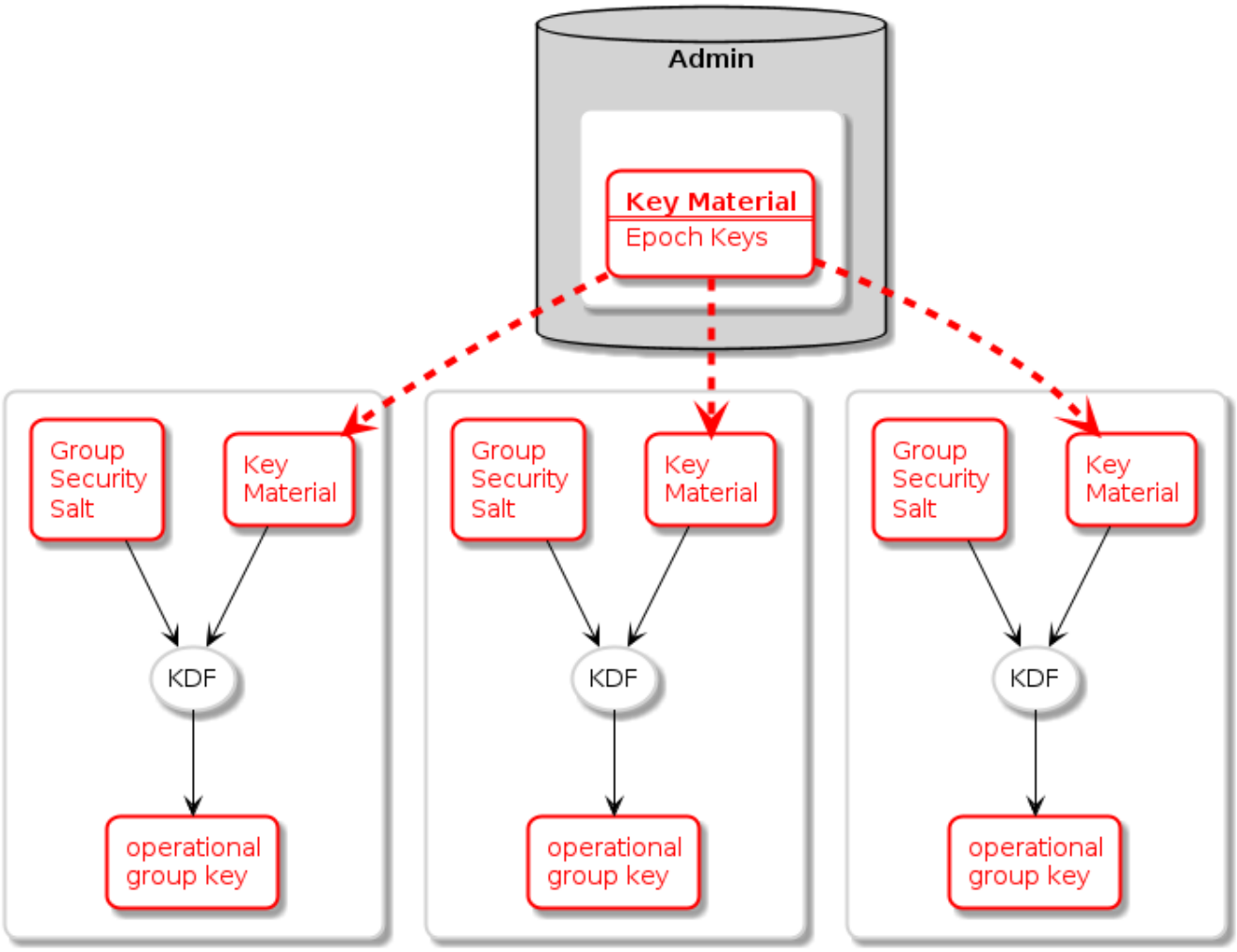


Figure 19. Group Key Distribution

4.15.4.1. Installing a Group onto a Newly Commissioned Node

This section provides an example of the operations required to install a group onto a newly com! missioned node.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

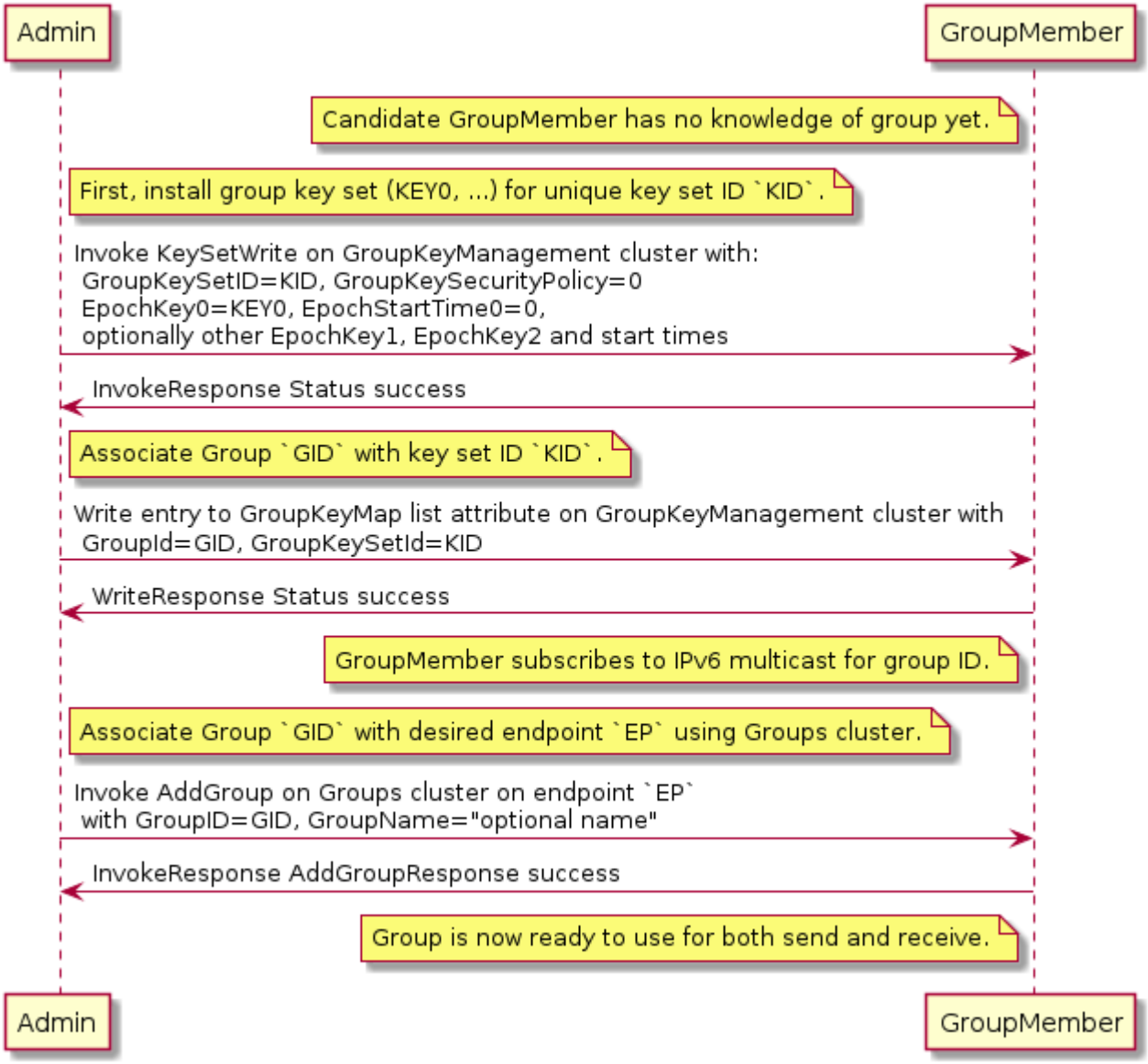


Figure 20. Installing a group onto a new node

Sequence:

¥ Admin :

%Generate fabric-unique group ID GIDand random key key0 for group key set ID K.

%Write the group key set K to the remote node, GroupMember, using KeySetWrite command.

%Associate group ID GIDwith key set K by writing an entry to the GroupKeyMap list attribute.

¥ GroupMember :

%Node subscribes to the IPv6 multicast address generated from the fabric ID and group ID.

¥ Admin :

%Associate endpoint with group ID GID by sending the Groups clusterÕs AddGroupcommand to endpoint.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

4.16. Message Counter Synchronization Protocol (MCSP)

This section describes the protocol used to securely synchronize the message counter used by a sender of messages encrypted with a symmetric group key.

Message counter synchronization is an essential part of enabling secure messaging between mem! bers of an operational group. Specifically, it protects against replay attacks, where an attacker replays older messages, which may result in unexpected behaviour if accepted and processed by the receiver.

The recipient of a message encrypted with a group key can trust and process that message only if it has kept the last message counter received from a given sender using that key.

Underlying MCSP is a simple request / response protocol. When a multicast message with unknown counter is received, synchronization via MCSP begins by sending a synchronization request via uni! cast UDP to the multicast message originatorÕs unicast IPv6 address. That originator then sends a synchronization response to the unsynchronized node via unicast UDP. After cryptographic verifi! cation, the formerly unsynchronized node is now synchronized with the originatorÕs message counter and can trust the original and subsequent messages from the originator node.

4.16.1. Message Counter Synchronization Methods

There are two methods for synchronizing the message counter of a peer node, which are config! urable per-group-key based on the GroupKeySecurityPolicy field of a given group key set (see GroupKeySetStruct).

4.16.1.1. Trust-first policy

The first authenticated message counter from an unsynchronized peer is trusted, and its message counter is used to configure message-counter-based replay protection on future messages from that node. All control messages (any message with C Flag set) use the control message counter and SHALL use Trust-first for synchronization. Note that MCSP is not used for Trust-first synchroniza! tion.

This policy provides lower latency for less security-sensitive applications such as lighting.

WARNING	Trust-first synchronization is susceptible to accepting a replayed message after a Node has been rebooted.

4.16.1.2. Cache-and-sync policy

The message that triggers message counter synchronization is stored, a message counter synchro! nization exchange is initiated, and only when the synchronization is completed is the original mes! sage processed. Cache-and-sync provides replay protection even in the case where a Node has been rebooted, at the expense of higher latency.

Support for the cache-and-sync policy and MCSP is optional. A node indicates its ability to support this feature via the Group Key Management cluster FeatureMap .

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

4.16.2. Group Peer State

The Group Peer State Table stores information about every peer with which the node had a group message exchange. For every peer node id the following information is available in the table:

- ¥ PeerÕs Encrypted Group Data Message Counter Status:
  - %Synchronized Data Message Counter - the largest encrypted data message counter received from the peer, if available.
  - %Flag to indicate whether this counter value is valid and synchronized.
  - %The message reception state bitmap tracking the recent window of data message counters received from the peer.
- ¥ PeerÕs Encrypted Group Control Message Counter Status:
  - %Synchronized Control Message Counter - the largest encrypted control message counter received from the peer, if available.
  - %Flag to indicate whether this counter value is valid and synchronized.
  - %The message reception state bitmap tracking the recent window of control message counters received from the peer.

There are three scenarios where the receiver of an encrypted message does not know the senderÕs last message counter:

- ¥ The encrypted message is the first received from a peer.
- ¥ The device rebooted without persisting the Group Peer State Table content. Note: it is not required to persist the Peer State Table.
- ¥ The entry for the Peer in the Group Peer State Table was expunged due to the table being full.

The next sections describe the functional protocol used to request message counter synchronization with a peer and form responses to such message counter synchronization requests.

4.16.3. MCSP Messages

4.16.3.1. MsgCounterSyncReq - Message Counter Synchronization Request

The Message Counter Synchronization Request (MsgCounterSyncReq) message is sent when a mes! sage was received from a peer whose current message counter is unknown.

A MsgCounterSyncReq message SHALL set the C Flag in the message header. The control message counter SHALL be used for message protection.

A MsgCounterSyncReq message SHALL be secured with the group key for which counter synchro! nization is requested and SHALL set the Session Type to 1, indicating a group session as per the rules outline in Section 4.16.5, ÒMessage Counter Synchronization ExchangeÓ .

The payload of the MsgCounterSyncReq message takes the format defined in Table 24, ÒMessage Counter Sync RequestÓ:

Table 24. Message Counter Sync Request

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Field Size	Message Field	Description
8 bytes	Challenge	The Challenge is a 64-bit random number generated using the DRBG by the initiator of a MsgCounterSyncReq to uniquely identify the synchronization request crypto! graphically.

4.16.3.2. MsgCounterSyncRsp - Message Counter Synchronization Response

The Message Counter Synchronization Response (MsgCounterSyncRsp) message is sent in response to a MsgCounterSyncReq.

A MsgCounterSyncRsp message SHALL set the C Flag in the message header. The control message counter SHALL be used for message protection.

The MsgCounterSyncRsp message has the format defined in Table 25, ÒMessage Counter Sync ResponseÓ.

Table 25. Message Counter Sync Response

Field Size	Message Field	Description
4 bytes	Synchronized Counter	The current data message counter for the node sending the MsgCounterSyncRsp message.
8 bytes	Response	The Response SHALL be the same as the 64-bit value sent in the Challenge field of the corresponding MsgCounter! SyncReq.

4.16.4. Unsynchronized Message Processing

An unsynchronized message is one that is cryptographically verified from a node whose message counter is unknown. Upon receipt of an unsynchronized message process the message as follows:

1. The message SHALL be of Group Session Type , otherwise discard the message.
2. If C Flag is set to 1:
   a. Create a Message Reception State and set its max_message_counter to the message counter of the given message, i.e. trust-first .
   b. Accept the message for further processing.
3. If C Flag is set to 0:
   a. Determine the Section 11.2.6.2.2, ÒGroupKeySecurityPolicyÓ (as set by the Section 11.2, ÒGroup Key Management ClusterÓ) of the operational group key used to authenticate the message.
   b. If the key has a trust-first security policy, the receiver SHALL:
      i. Set the peerÕs group key data message counter to Message Counter of the message.
         A. Clear the Message Reception State bitmap for the group session from the peer.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

B. Mark the peer 's group key data message counter as synchronized.

ii. Process the message.

c. If the key has a cache-and-sync security policy, the receiver SHALL:

i. If MCSP is not in progress for the given peer Node ID and group key:

A. Store the message for later processing.

B. Proceed to Section 4.16.5, ÒMessage Counter Synchronization ExchangeÓ .

ii. Otherwise, do not process the message.

A. An implementation MAY queue the message for later processing after MCSP com! pletes if resources allow.

For each peer Node ID and group key pair there SHALL be at most one synchronization exchange outstanding at a time.

4.16.5. Message Counter Synchronization Exchange

A message synchronization exchange starts by sending the MsgCounterSyncReq message to the peer Node that sent the message with unknown message counter. When a synchronization request is triggered by an incoming multicast message, the Node SHALL first wait for a uniformly random amount of time between 0 and MSG_COUNTER_SYNC_REQ_JITTER.

The sender of the MsgCounterSyncReq message SHALL:

1. Set the Message Header fields as follows:
   a. The S Flag SHALL be set to 1.
   b. The DSIZ field SHALL be set to 1.
   c. The P Flag SHALL be set to 1.
   d. The C Flag SHALL be set to 1.
   e. The Session Type field SHALL be set to 1.
   f. The Session ID field SHALL be set to the Group Session Id for the operational group key being synchronized.
   g. The Source Node ID field SHALL be set to the Node ID of the sender Node.
   h. The Destination Node ID field SHALL be set to the Source Node ID of the message that trig! gered the synchronization attempt.
2. Create a new synchronization Exchange .
   a. The Exchange ID of the message SHALL be set to match the new Exchange.
   b. The I Flag SHALL be set to 1.
   c. The A Flag SHALL be set to 0.
   d. The R Flag SHALL be set to 1.
3. Set the Challenge field to the value returned by Crypto_DRBG(len = 8 * 8) and store that value to resolve synchronization responses from the destination peer.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

4. Arm a timer to enforce that a synchronization response is received before MSG_COUNTER_!SYNC_TIMEOUT.
   a. Upon firing of the timer:
      i. The synchronization exchange SHALL be closed.
      ii. Any message waiting on synchronization associated with the exchange SHALL be dis!carded.
   b. The timer SHALL be stopped upon receipt of an authenticated MsgCounterSyncRsp message that matches:
      i. The Source Node ID field matches the Destination Node ID field of the most recent Msg!CounterSyncReq message generated for that Node.
      ii. The Response field corresponds to the Challenge field of the MsgCounterSyncReq mes!sage.
5. Invoke Section 4.6.1, ÒMessage TransmissionÓ using parameters from step 1 to encrypt and then send the request message over UDP to the IPv6 unicast address of the destination.
   a. The request message SHALL use the same operational group key as the message which trig!gered synchronization.
   b. The group key SHALL be known/derivable by both parties (sender and receiver).

The receiver of MsgCounterSyncReq SHALL:

1. Verify the Destination Node ID field SHALL match the Node ID of the receiver, otherwise discard the message.
2. Respond with MsgCounterSyncRsp.

The sender of the MsgCounterSyncRsp response message SHALL:

1. Set the Message Header fields as follows:
   a. The S Flag SHALL be set to 1.
   b. The DSIZ field SHALL be set to 1.
   c. The P Flag SHALL be set to 1.
   d. The C Flag SHALL be set to 1.
   e. The Session Type field SHALL be set to 1.
   f. The Session ID field SHALL be set to the Group Session Id for the operational group key being synchronized.
   g. The Source Node ID field SHALL be set to the Node ID of the sender Node.
   h. The Destination Node ID field SHALL be set to the Source Node ID of the MsgCounterSyn!cReq.
2. Set the MsgCounterSyncRsp payload fields as follows:
   a. The Response field SHALL be set to the value of the Challenge field from the MsgCounterSyn!cReq.
   b. The Synchronized Counter field SHALL be set to the current Global Group Encrypted Data

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Message Counter of the sender.

3. Use the same exchange context as the MsgCounterSyncReq being responded to.
   a. The Exchange ID of the message SHALL be set to the Exchange ID of the MsgCounterSyncReq.
   b. The I Flag SHALL be set to 0.
   c. The A Flag SHALL be set to 1.
   d. The R Flag SHALL be set to 1.
4. Invoke Section 4.6.1, ÒMessage TransmissionÓ using parameters from step 1 to encrypt and then send the response message over UDP to the IPv6 unicast address of the destination.

The receiver of the MsgCounterSyncRsp message SHALL:

1. Verify the MsgCounterSyncRsp matches a previously sent MsgCounterSyncReq:
   a. An active synchronization exchange SHALL exist with the source node.
   b. The Exchange ID field SHALL match the Exchange ID used for the original MsgCounterSyncReq message.
   c. The Response field SHALL match the Challenge field used for the original MsgCounterSyncReq message.
   d. The Destination Node ID field SHALL match the Source Node ID of the original MsgCounterSyncReq message.
   e. The Source Node ID field SHALL match the Destination Node ID of the original MsgCounterSyncReq message.
2. On verification failure:
   a. Silently ignore the MsgCounterSyncRsp message.
3. On verification success:
   a. Set the peerÕs group key data message counter to Synchronized Counter .
   b. Clear the Section 4.5.4.1, ÒMessage Reception StateÓ bitmap for the peer.
   c. Mark the peer 's group key data message counter as synchronized.
   d. Resume processing of any queued message that triggered synchronization according to Section 4.5.6, ÒCounter Processing of Incoming MessagesÓ .
      i. If more than one message is queued from the synchronized peer, using the same operational group key, the messages SHALL be processed in the order received.
   e. Close the synchronization exchange created for the original MsgCounterSyncReq message.

4.16.6. Message Counter Synchronization Session Context

While conducting Message Counter Synchronization with a peer, nodes SHALL maintain the following session context. For nodes initiating message counter synchronization, this context SHALL be maintained throughout the full exchange of MsgCounterSyncReq and MsgCounterSyncRsp messages. For nodes responding to MsgCounterSyncReq messages, the context SHALL only be maintained long enough to generate and successfully transmit the MsgCounterSyncRsp message.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

1. Fabric Index : Records the Index for the Fabric within which nodes are conducting message counter synchronization.

   %Fabric Index is derived by identification of an Operational Group Key associated with the fabric through successful decryption with that key and verification of the Message Integrity Check. For nodes initiating counter synchronization, this occurs at decryption of an inbound groupcast message. For nodes in the responder role, this occurs at decryption of an inbound MsgCounterSyncReq message.

2. Peer Node ID: Records the node ID of the peer with which message counter synchronization is being conducted.

   %For nodes initiating message counter synchronization, this is the node ID of the responder. For nodes responding to message counter synchronization, this is the node ID of the initia! tor.

3. Role: Records whether the node is the initiator of or responder to message counter synchroniza! tion.

   %Together, Fabric Index , Peer Node ID and Role comprise a unique key that can be used to match incoming messages to ongoing MCSP exchanges.

4. Message Reception State: Provides tracking for the Control Message Counter of the remote peer.
5. Peer IP Address : The unicast IP address of the peer.
6. Peer Port : The receiving port of the peer.
7. Operational Group Key : The Operational Group Key that is being used to encrypt messages within the counter synchronization exchange.
8. Group Session ID: Records the Group Session ID derived from the Operational Group Key that is being used to encrypt messages within the counter synchronization exchange.

4.16.7. Sequence Diagram

The following sequence diagram shows an example of how message counter synchronization behaves in the most common scenario.

4.16.7.1. Scenario 1"Ñ"Multicast Receiver Initiated

Assumptions:

¥ Sender transmits a multicast message.

¥ Receiver does not know Sender 's message counter.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

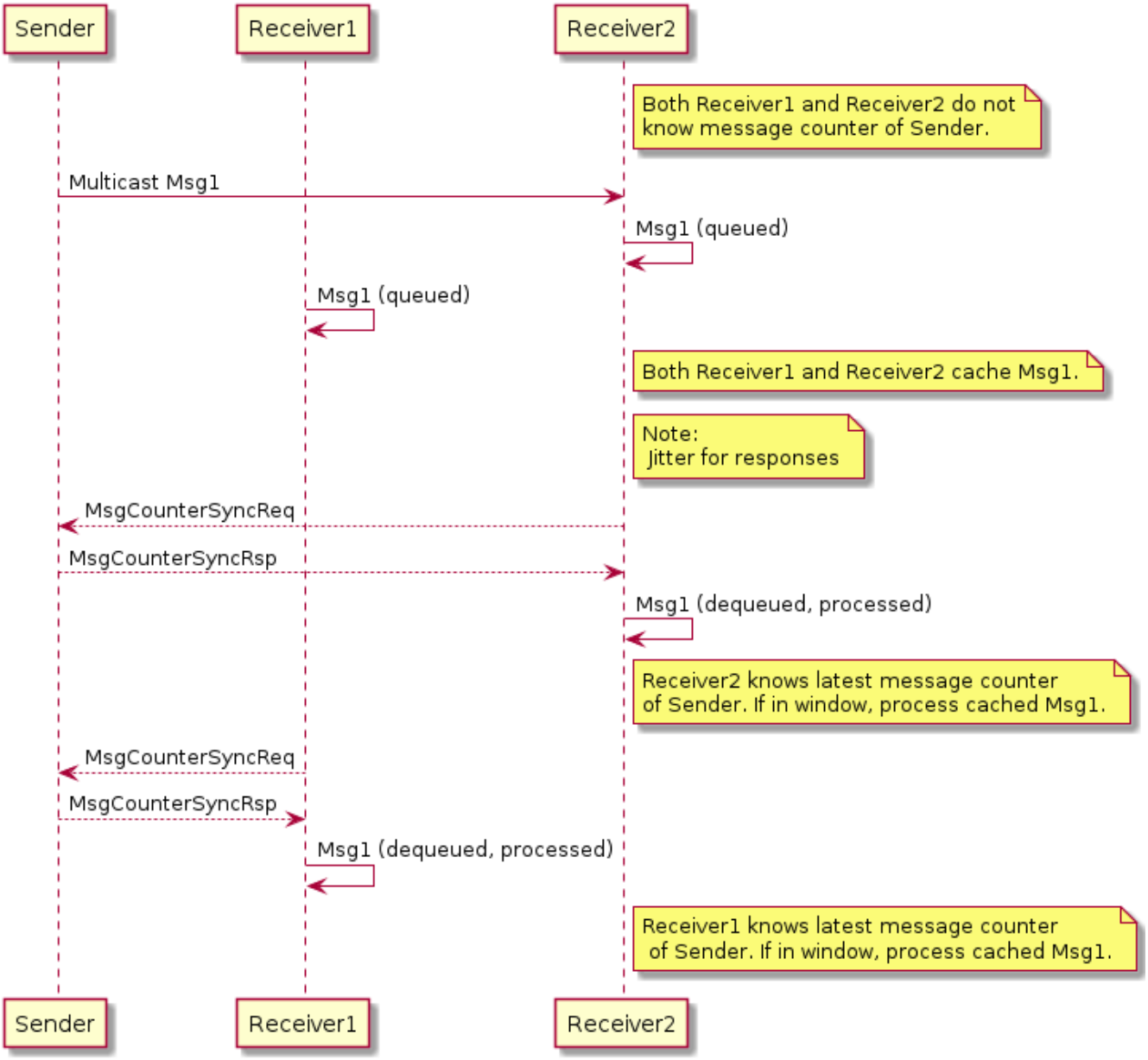


Figure 21. Multicast Receiver Initiated Message Counter Synchronization

Sequence:

¥ Sender:

%Generates, encrypts and sends Msg1 as a multicast message. Msg1 could be:

& Regular message that starts encrypted group communication with receivers Receiver1 and Receiver2.

¥ Receivers Receiver1 and Receiver2 each:

%Receive, decrypt, authenticate, and cache Msg1 message for later processing.

& Generate, encrypt, and send MsgCounterSyncReq message.

¥ Sender:

%Receives MsgCounterSyncReq message.

%Generates, encrypts and sends MsgCounterSyncRsp message.

¥ R1 and R2 each:

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

%Receive, decrypt, process, and verify MsgCounterSyncRsp message from Sender.

%On verification success: marks Sender 's group key message counter as synchronized.

& Processes cached Msg1 message.

4.17. Bluetooth Transport Protocol (BTP)

The Bluetooth Transport Protocol (BTP) provides a TCP-like layer of reliable, connection-oriented data transfer on top of GATT. BTP splits individual Service Data Unit (SDU) messages into multiple BTP segments, which are each transmitted via a single GATT write or indication (as shown in Figure 22, ÒMATTERoBLE: Matter Message / BTP layeringÓ).

While BTP is a generic protocol, Matter specifically uses BTP to define a Matter-over-Bluetooth Low Energy (MATTERoBLE) Interface. A MATTERoBLE Interface MUST implement BTP as a universally compatible transport mode. A MATTERoBLE Interface SHALL only be used to transport Matter mes! sages as the BTP SDU.

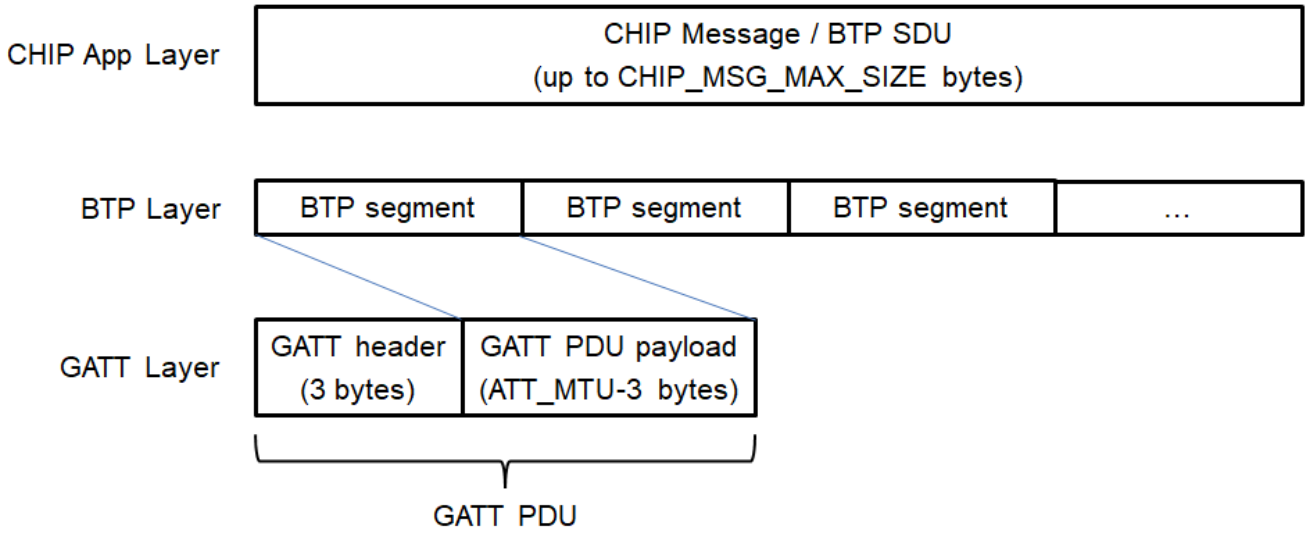


Figure 22. MATTERoBLE: Matter Message / BTP layering

The BTP session handshake allows devices to check BTP protocol version compatibility and exchange other data before a BTP session is established. Once established, this session is used to send and receive BTP SDUs (such as Matter messages) as BTP Message Segments. A BTP session MAY open and close with no effect on the state of the underlying Bluetooth LE connection, except in the case where a BTP session is closed by the Peripheral Device. Either the Peripheral or the Central MAY signal the end of a BTP session by closing the underlying BLE connection.

Due chiefly to constraints put on design by resource-limited BLE chipsets, BTP defines a receive window for each side of a session in units of GATT PDUs. Each GATT Write Characteristic Value (ATT_WRITE_REQ) PDU or Indication (ATT_HANDLE_VALUE_IND) PDU is sent with a sequence num! ber which the receiver uses to acknowledge receipt of each packet at the BTP layer and open its receive window from the senderÕs perspective.

4.17.1. BTP Session Interface

Conceptually, an open BTP session is exposed to the next-higher session layer as a full-duplex mes!

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

sage stream.

4.17.2. BTP Frame Formats

A BTP Frame consists of an 8-bit header followed by one or more optional fields, as detailed below. BTP uses little endian encoding for any header fields larger than one byte in length.

4.17.2.1. BTP Packet Protocol Data Unit

Table 26, ÒBTP Packet PDU formatÓdefines the BTP Packet PDU format.

Unused fields SHALL be set to '0'.

Table 26. BTP Packet PDU format

bit 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
-	H	M	-	A	E	-	B	[Management Opcode]							
[Ack Number]								[Sequence Number]							
[Message Length]															
[Segment Payload]É															
É															

H (Handshake) bit

Set to '0' for normal BTP packets. When set, this bit indicates a BTP handshake packet for session establishment and has a different packet format described below.

M (Management Message) bit

Indicates the presence ('1') or absence ('0') of the Management Opcode field. All segments of a mes! sage MUST set this bit to the same value.

A (Acknowledgement) bit

Indicates the presence of the Ack Number field.

B (Beginning Segment) bit

Set to '1' on the first segment of a BTP SDU and set to '0' for all remaining segments of the same BTP SDU. It indicates the presence of the Message Length field.

E (Ending Segment) bit

Set to '1' on the last segment of a BTP SDU and set to '0' for all other segments of the same BTP SDU. A segment MAY have both the Beginning and Ending bits set indicating that a full BTP SDU is included in the message. When set, the segment payload length is equal to the total remaining unre! ceived message data. When not set, the segment payload length is equal to the maximum allowable BTP session packet size minus header overhead.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Ack Number

Optional field specified in Section 4.17.3.8, ÒPacket AcknowledgementsÓ.

Sequence Number

Mandatory field for regular data messages specified in Section 4.17.3.6, ÒSequence NumbersÓ.

Message Length

Optional field present in Beginning Segment only. Value indicates the length in bytes of the full mes! sage buffer to be transmitted. None of the BTP Packet PDU fields is included in the Message Length.

Segment Payload

Optional field containing a segment of the Service Data Unit (SDU) message in transmission to the receiver.

4.17.2.2. BTP Control Frames

BTP defines different control frame formats depending on the Management Opcode that is in the BTP Packet PDU header. Valid Management Opcodes for BTP Control Frames are defined in Table 27, ÒBTP Control codesÓ.

Table 27. BTP Control codes

Management Opcode	Name	Description
0x6C	Handshake	Request and response for BTP session establishment

4.17.2.3. BTP Handshake Request

Table 28. BTP Handshake Request format

bit 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	H=1	M=1	0	A=0	E=1	0	B=1	Management Opcode = 0x6C							
Ver[1]				Ver[0]				Ver[3]				Ver[2]			
Ver[5]				Ver[4]				Ver[7]				Ver[6]			
Requested ATT_MTU															
Client Window Size															

H (Handshake) bit

Set to '1' for connection handshake messages.

M (Management) bit

Set to '1' for connection handshake messages.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Ver[i] (Version) nibbles

Used to negotiate the highest version capability between a Device pair. Supported versions are listed once each, newest first, in descending order. Unused version fields are filled with Ô0Õ.

The following values are defined:

¥ 0$Ñ$Unused field

¥ 4$Ñ$BTP as defined by Matter v1.0

Requested ATT_MTU

Requested ATT_MTU is a 16-bit unsigned integer field containing the size of the GATT PDU (ATT_MTU) that can be received by the sender minus the size of the GATT header (3). This value is obtained via the standard ATT MTU exchange procedure (see Bluetooth¨ Core Specification 4.2 Vol 3, Part F, Section 3.4.2 "MTU Exchange") and is used to validate that both sides of the BLE connec! tion are using the common minimum value. If BTP is not aware of the negotiated GATT MTU, the value shall be set to '23', indicating the minimum ATT_MTU defined by GATT. If the client has no preference, the value may be set to '0'.

NOTE	Each GATT PDU used by the BTP protocol introduces 3 byte header overhead mak! ing the maximum BTP Segment Size for the session equal to negotiated ATT_MTU-3.

Client Window Size

Value of the maximum receive window size supported by the server, specified in units of BTP pack! ets where each packet may be up to 244 bytes in length. This maximum was chosen so a single BTP segment can fit into a single 255 byte BLE link layer PDU, including all headers from the link layer, L2CAP, GATT, and BTP.

4.17.2.4. BTP Handshake Response

Table 29. BTP Handshake Response format

bit 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	H=1	M=1	0	A=0	E=1	0	B=1	Management Opcode = 0x6C							
Reserved				Final Protocol Version				Selected ATT_MTU (low byte)É							
ÉSelected ATT_MTU (high byte)								Selected Window Size							

H (Handshake) bit

Set to '1' for session handshake messages.

M (Management) bit

Set to '1' for session handshake messages.

Reserved

Must be set to '0'.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Final Protocol Version

Value of the BTP protocol version selected by the server.

Selected ATT_MTU

Value of the maximum ATT_MTU for the connection selected by the server as a 16-bit unsigned inte! ger.

Selected Window Size

Value of the maximum receive window size supported by the server, specified in units of BTP pack! ets where each packet may be up to the selected segment size in length.

4.17.3. BTP GATT Service

4.17.3.1. BTP Channelization

Bluetooth Transport Protocol provides a packetized stream interface over GATT but says nothing about the actual contents of the data packets it transports. The BTP Service UUID is used to specify the actual contents of the packets (see Table 30, ÒBTP Service UUIDÓ).

Table 30. BTP Service UUID

BTP Datagram Contents	BTP Service UUID
Matter Message frames	MATTER_BLE_SERVICE_UUID Note: See Section 4.17.5, ÒBluetooth SIG Consid! erationsÓ for terms of use.

While a single BTP connection may exist between a BTP Client and BTP Server, multiple BTP ses! sions may be established between various peers.

4.17.3.2. BTP GATT Service

The BTP GATT service is composed of a service with three characteristicsÑC1, C2 and C3 (see Table 31, ÒBTP GATT serviceÓ). The client SHALL exclusively use C1 to initiate BTP sessions by sending BTP handshake requests and send data to the server via GATT ATT_WRITE_CMD PDUs. While a client is subscribed to allow indications over C2, the server SHALL exclusively use C2 to respond to BTP handshake requests and send data to the client via GATT ATT_HANDLE_VALUE_IND PDUs.

Table 31. BTP GATT service

Attribute	Description
BTP Service	UUID = MATTER_BLE_SERVICE_UUID
C1 Characteristic (Client TX Buffer)	UUID = 18EE2EF5-263D-4559-959F-4F9C429F9D11 Characteristic Porperties = Write max length = 247 bytes
C2 Characteristic (Client RX Buffer)	UUID = 18EE2EF5-263D-4559-959F-4F9C429F9D12 Characteristic Properties = Indication max length = 247 bytes

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Attribute	Description
C3 Characteristic (Additional commissioning-related data)	UUID = 64630238-8772-45F2-B87D-748A83218F04 Characteristic Properties = Read max length = 512 bytes

For all messages sent from the BTP Client to BTP Server, BTP SHALL use the GATT Write Characteristic Value sub-procedure. For all messages sent from the BTP Server to BTP Client, BTP SHALL use the GATT Indications sub-procedure.

The values of C1 and C2 SHALL both be limited to a maximum length of 247 bytes. This constraint is imposed to align with maximum PDU size when LE Data Packet Length Extensions (DPLE) is enabled on Bluetooth 4.2 hardware. Per Bluetooth¨ Core Specification 4.2 Vol 3, Part F, Section 3.2.9 "Long Attribute Values", the maximum characteristic value length is 512 bytes. The maximum lengths of C1 and C2 may increase in a future version of the BTP specification to allow higher throughput on BLE connections whose ATT_MTU exceeds 247 bytes.

C3 is an optional characteristic that the server MAY use to include additional data required during the provisioning as defined in Section 5.4.2.5.7, ÒGATT-based Additional DataÓ.

BTP Clients SHALL perform certain GATT operations synchronously, including GATT discovery, subscribe, and unsubscribe operations. GATT discovery, subscribe, or unsubscribe SHALL NOT be initiated while the result of a previous operation remains outstanding. This requirement is imposed by GATT protocol.

4.17.3.3. Session Establishment

Before a BTP session can be initiated, a central SHALL first establish a BLE connection to a peripheral. Once a BLE connection has been formed, centrals SHALL assume the GATT client role for BTP session establishment and data transfer, and peripherals SHALL assume the GATT server role. If peripheral supports LE Data Packet Length Extension (DPLE) feature it SHOULD perform data length update procedure before establishing a GATT connection.

Before establishing a BTP session, the GATT client SHOULD perform a GATT Exchange MTU procedure.

After that the BTP Client SHALL execute the GATT discovery procedure. The GATT discovery procedure starts with primary service discovery using either the GATT Discover All Primary Services sub-procedure or the GATT Discover Primary Services by Service UUID sub-procedure.

The BTP Client SHALL perform either the GATT Discover All Characteristics of a Service sub-procedure or the GATT Discover Characteristics by UUID sub-procedure in order to discover the C1 and C2 characteristics.

The BTP Client SHALL perform the GATT Discover All Characteristic Descriptors sub-procedure in order to discover the Client Characteristic Configuration descriptor (CCCD) of C2 characteristic.

To initiate a BTP session, a BTP Client SHALL send a BTP handshake request packet to the BTP Server via a ATT_WRITE_CMD PDU on characteristic C1 of the BTP Service. The handshake request packet SHALL include, a list of BTP protocol versions supported by the client, the clientÕs GATT

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

ATT_MTU, and the clientÕs maximum receive window size. The list of supported protocol versions SHALL be sorted in descending numerical order. If the client cannot determine the BLE connec! tionÕs ATT_MTU, it SHALL specify a value of '23' (the minimum ATT_MTU supported by GATT) for this field in the handshake request. For a detailed specification of the handshake request binary data format, see Section 4.17.2.3, ÒBTP Handshake RequestÓ.

After the BTP Client acknowledges delivery of the handshake request packet, upon receipt of a GATT Write Response, the BTP Client SHALL enable indications over C2 characteristics by writing value 0x01 to C2Õs Client Characteristic Configuration descriptor as described in Bluetooth¨ Core Specification 4.2 Vol 3, Part C, Section 10.3.1.1 "Handling GATT Indications and Notifications".

Once the GATT server has received a clientÕs BTP handshake request and confirmed the clientÕs sub! scription to C2, it SHALL send a BTP handshake response to the client via a ATT_HANDLE_VAL! UE_IND PDU on C2. This response SHALL contain the window size, maximum BTP packet size, and BTP protocol version selected by the server. For a detailed specification of the handshake response binary data format, see Section 4.17.2.4, ÒBTP Handshake ResponseÓ.

The server SHALL select a window size equal to the minimum of its and the clientÕs maximum win! dow sizes. Likewise, the server SHALL select a maximum BTP Segment Size for the BLE connection by taking the minimum of 244 bytes (the maximum characteristic value length of C1 and C2), serverÕs ATT_MTU-3 and ATT_MTU-3 as declared by the client.

The server SHALL select a BTP protocol version that is the newest which it and the client both sup! port, where newer protocol version numbers are strictly larger than those of older versions. The version number returned in the handshake response SHALL determine the version of the BTP pro! tocol used by client and server for the duration of the session.

If the server determines that it and the client do not share a supported BTP protocol version, the server SHALL close its BLE connection to the client. When a client sends a handshake request, it SHALL start a timer with a globally-defined duration of BTP_CONN_RSP_TIMEOUT seconds. If this timer expires before the client receives a handshake response from the server, the client SHALL close the BTP session and report an error to the application. Likewise, a server SHALL start a timer with the same duration when it receives a handshake request from a client. If this timer expires before the server receives a subscription request on C2, the server SHALL close the BTP session and report an error to the application. The state machine in Figure 23, ÒBTP session handshakeÓ illus! trates the function of these timers as part of the BTP session establishment procedure.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

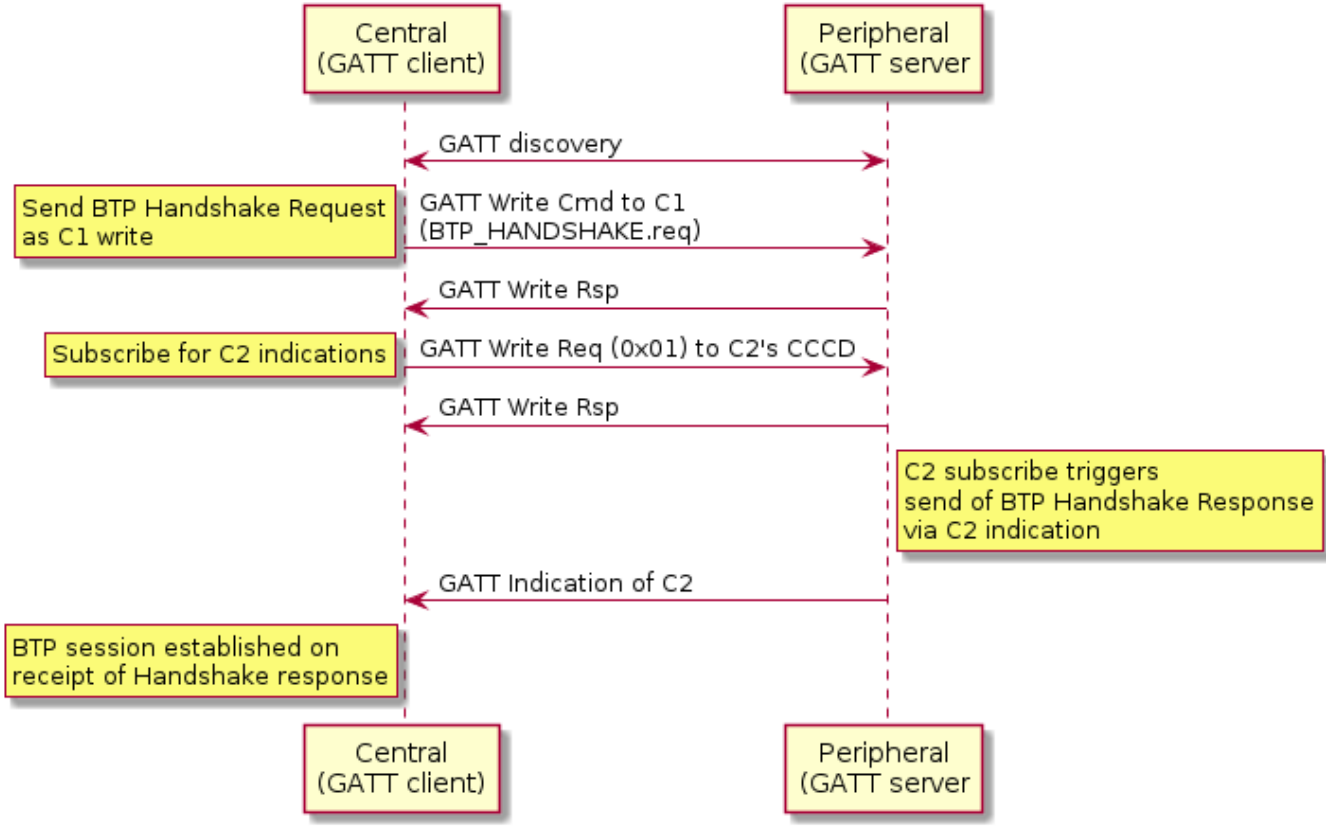


Figure 23. BTP session handshake

4.17.3.4. Data Transmission

Once a BTP session has been established, the next-higher-layer application on both peers may use this BLE connection to send and receive data via GATT writes or indications, according to a peerÕs GATT role. Clients SHALL exclusively use GATT Write Characteristic Value sub-procedure to send data to servers and servers SHALL exclusively use GATT Indication sub-procedure to send data to clients.

All BTP packets sent on an open BLE connection SHALL adhere to the BTP Packet PDU binary data format specified in BTP Frame Formats. All BTP packets SHALL include a header flags byte and an 8-bit unsigned sequence number. All other packet fields are optional. These optional fields include an 8-bit unsigned received packet acknowledgement number, 16-bit unsigned buffer length indica! tion, and variable-length buffer segment payload.

This section is defined entirely within the scope of a single BTP session. Concurrent BTP sessions between the same peer and multiple remote hosts SHALL maintain separate and independent acknowledgement timers, sequence numbers, and receive windows.

4.17.3.5. Message Segmentation and Reassembly

When the session layer (that is, MATTERoBLE) sends a Matter Message as a BTP SDU over a BTP ses! sion, that BTP SDU SHALL be split into ordered, non-overlapping BTP segments so the set of all BTP segments may be reassembled into the original BTP SDU (see Figure 22, ÒMATTERoBLE: Matter Mes! sage / BTP layeringÓ). Each BTP segment SHALL be prepended with a BTP packet header and sent as the payload of a single GATT PDU. If a BTP SDU is split into more than one BTP segment, the BTP segments SHALL be sent in order of their position in the original BTP SDU, starting with the BTP segment at the bufferÕs head.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

At any point in time, only one BTP SDU may be transmitted in each direction over a BTP session. The transmission of BTP segments of any two BTP SDUs SHALL not overlap. If the application attempts to send one BTP SDU while transmission of another BTP SDU is in progress, the new BTP SDU SHALL be appended to a first-in, first-out queue. The next BTP SDU SHALL be dequeued from this queue and transmitted once transmission of the current BTP SDU completes, that is, once all BTP segments of the current BDP SDU have been transmitted and received by the peer via GATT.

The first BTP segment of a BTP SDU sent over a BTP session SHALL have its Beginning Segment header flag set to indicate the beginning of a new BTP SDU (see Table 26, ÒBTP Packet PDU formatÓ). The presence of this flag SHALL indicate the further presence of a 16-bit unsigned integer field, the Message Length, that provides the receiver with the total length of the BTP SDU. The last BTP segment for a given BTP SDU SHALL have its Ending Segment flag set to indicate the end of the transmitted BTP SDU. A BTP packet that bears an unsegmented BTP SDUÑthat is, a BTP SDU small enough to fit into a single BTP segmentÑSHALL have both its Beginning Segment and Ending Segment flags set.

The size of a single BTP SDU sent via BTP is limited to 64KB, that is, the maximum size of the Message Length field in the BTP packet header. The number of segments used to send a buffer is unlimited and delimited by the Beginning Segment and Ending Segment bits in the BTP packet header. The upper layer imposes more stringent requirements over the maximum SDU size, such as Section 4.4.4, ÒMessage Size RequirementsÓ

The length of the data payload in each BTP segment whose Ending Segment bit is not set SHALL be equal to the sessionÕs maximum BTP packet size minus the size of that packetÕs header. If a packetÕs Ending Segment bit is set, the length of its BTP segment data payload SHALL equal the size of the original BTP SDU minus the total size of all previously transmitted BTP segments of that BTP SDU. In this way, the length of a SDUÕs last BTP segment is implied by its size.

Once a peer receives a complete set of BTP segments, it SHALL reassemble them in the order received, and verify that the reassembled BTP SDUÕs total length matches that specified by the Beginning SegmentÕs Message Length value. If they match, the receiver SHALL pass the reassembled BTP SDU up to the next-higher-layer. If the reassembled bufferÕs length does not match that specified by the sender, or if received BTP segment payload size would exceed the maximum BTP packet size, or receiver receives an Ending Segment without the presence of a previous Beginning Segment, or a Beginning Segment when another BTP SDUÕs transmission is already in progress, the receiver BTP SHALL close the BTP session and report an error to the application.

4.17.3.6. Sequence Numbers

All BTP packets SHALL be sent with sequence numbers, regardless of whether they contain SDU segments (for example, a packet acknowledgement with no attached segment payload). The purpose of sequence numbers is to facilitate the BTP receive window. A BTP sequence number SHALL be defined as an unsigned 8-bit integer value that monotonically increments by 1 with each packet sent by a given peer. A sequence number incremented past 255 SHALL wrap to zero.

Sequence numbers SHALL be separately defined for either direction of a BTP session. The sequence number of the first packet sent by the client after completion of the BTP session handshake SHALL be zero. The serverÕs BTP handshake response bears an implied sequence number of zero because it occupies a slot in the clientÕs receive window. The client acknowledges the serverÕs BTP hand!

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

shake response with an acknowledgement sequence of zero. For this reason, the sequence number of the first data packet sent by the server after completion of the BTP session handshake SHALL be 1.

Peers SHALL check to ensure that all received BTP packets properly increment the senderÕs previ! ous sequence number by 1. If this check fails, the peer SHALL close the BTP session and report an error to the application.

4.17.3.7. Receive Windows

The purpose of the receive window is to enable flow control at the GATT session layer between BTP peers.

Flow control is required at the GATT transport layer for embedded platforms that use "minimal" BLE chipsets. These platforms may have limited space on the host processor to receive packets from their BLE chipsets. In the case of some dual-chip architectures, writes and indications are received and confirmed by the BLE chip with no input from the host processor. When the BLE chip sends the result of a received GATT PDU to the host processor, that payload and the corresponding BTP packet will be permanently lost if the host does not have enough space to receive it. For this reason, knowl! edge of a remote hostÕs ability to reliably receive GATT PDUs is presented at the transport layer in the form of the BTP receive window.

Both peers in a BTP session SHALL define a receive window, where the windowÕs size indicates the number of GATT PDUs (that is, BTP segments) a peer can reliably receive and store without session-layer acknowledgment. A maximum window size SHALL be established for both peers as part of the BTP session handshake. To prevent sequence number wrap-around, the largest maximum win! dow size any peer may support is 255.

Both peers SHALL maintain a counter to reflect the current size of the remote peerÕs receive win! dow. Each peer SHALL decrement this counter when it sends a packet via GATT write or indication and increment this counter when a sent packet is acknowledged.

If a local peerÕs counter for a remote peerÕs receive window is zero, the window SHALL be consid! ered closed, and the local peer SHALL NOT send packets until the window reopens (is incremented above zero). When a closed window reopens, a local peer SHALL immediately resume any pending BTP packet transmission.

A local peer SHALL also not send packets if the remote peerÕs receive window has one slot open and the local peer does not have a pending packet acknowledgement. This is to avoid the situation where the receive windows of both peers are full and neither can send an acknowledgement to reopen its window for the other. Because the serverÕs handshake response bears an implicit BTP sequence number of zero, a server SHALL initialize its counter for the clientÕs receive window size at (negotiated maximum window size - 1). A client SHALL initialize its counter for the serverÕs receive window at the negotiated maximum window size.

Both peers SHALL also keep a counter of their own receive window size based on the sequence number difference between the last packet they received and the last packet they acknowledged. This counter is used to proactively send early packet acknowledgements when a peerÕs own receive window is about to close. See Section 4.17.3.8, ÒPacket AcknowledgementsÓ for details.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

An example scenario involving BTP receive windows is depicted in Figure 24, ÒExample receive window scenarioÓ , complete with packet acknowledgements as specified in Section 4.17.3.8, ÒPacket AcknowledgementsÓ . In this scenario, the client transmits a three-segment buffer to the server once it receives the serverÕs handshake request. The handshake request occupies one slot in the clientÕs initial receive window. The serverÕs initial receive window is empty. Both client and server have a maximum window size of 4.

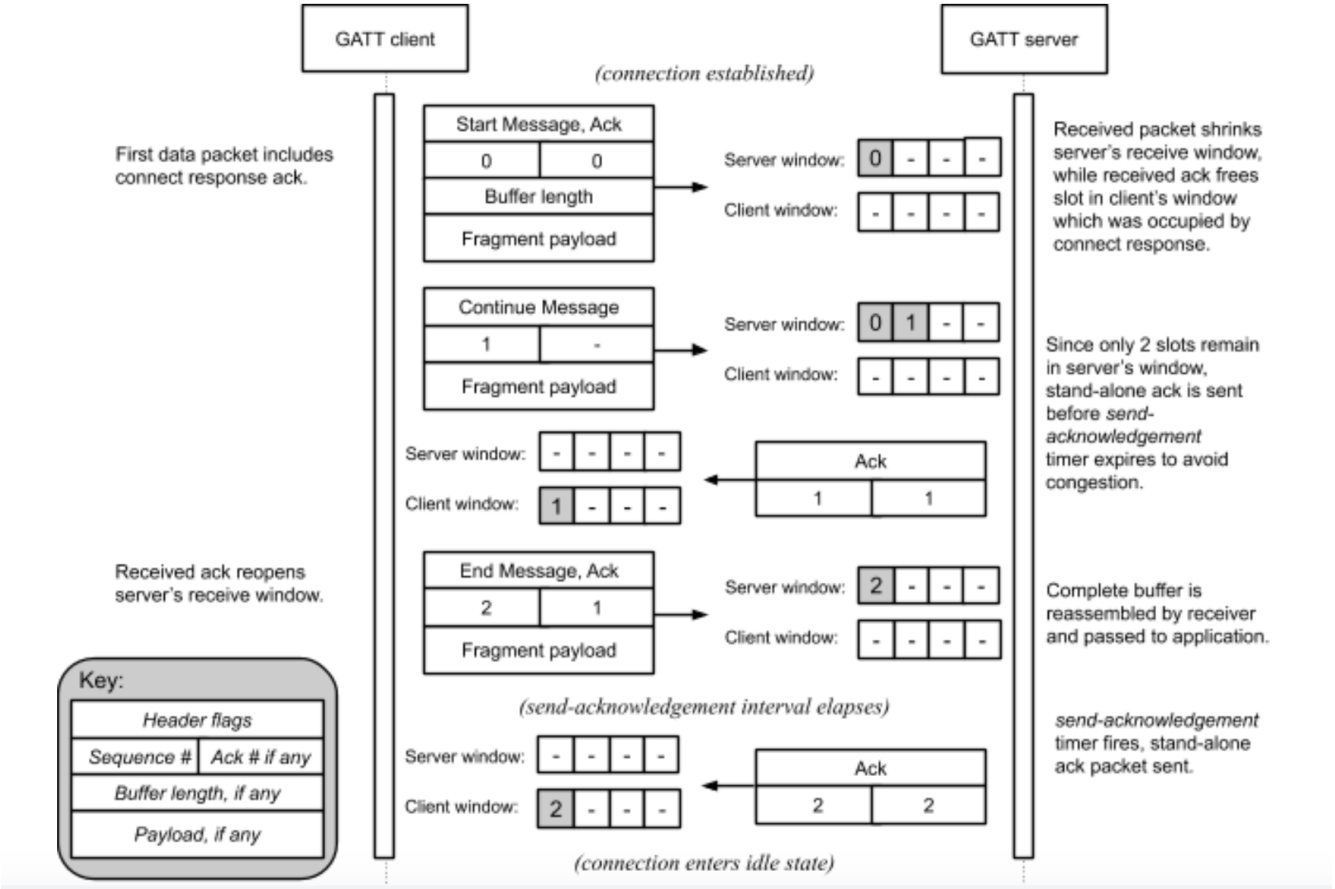


Figure 24. Example receive window scenario

4.17.3.8. Packet Acknowledgements

The purpose of sequence numbers and packet receipt acknowledgements is to support the BTP receive window and provide a keep-alive signal when a session is idle to affirm the health and con! tinued operation of a remote BTP stack.

Per BTP Frame Formats, BTP packet receipt acknowledgements SHALL be received as unsigned 8-bit integer values in the header of a BTP packet. The value of this field SHALL indicate the sequence number of the acknowledged packet.

Acknowledgement of a sequence number indicates acknowledgement of the previous sequence number, if it too is unacknowledged. By induction, acknowledgement of a given packet implies acknowledgement of all packets received on the same BTP session prior to the acknowledged packet.

An acknowledgement is invalid if the acknowledged sequence number does not correspond to an outstanding, unacknowledged BTP packet sequence number. In contrast to TCP, BTP acks are not "free." A stand-alone ackÑthat is, a BTP packet that contains a packet receipt acknowledgement

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

value but no buffer segment payloadÑconsumes a slot in a remote peerÕs window just like any other packet. Stand-alone acknowledgement packets SHALL be acknowledged by a remote peer. The implications of this are examined in Section 4.17.3.9, ÒIdle Connection StateÓ.

Each peer SHALL maintain an acknowledgement-received timer. When a peer sends any BTP packet, it SHALL start this timer if it is not already running. The timerÕs duration SHALL be globally defined as BTP_ACK_TIMEOUT seconds, referred to as the acknowledgement timeout interval.

A peer SHALL restart its acknowledgement-received timer when a valid acknowledgement is received for any but its most recently sent unacknowledged packet. A peer SHALL stop its acknowl! edgement-received timer if it receives an acknowledgement for its most recently sent unacknowl! edged packet. If a peerÕs acknowledgement-received timer expires, or if a peer receives an invalid acknowledgement, the peer SHALL close the BTP session and report an error to the application.

Because the serverÕs handshake response bears an implicit BTP sequence number of zero, a server SHALL start its acknowledgement-received timer when it sends a handshake response.

Each peer SHALL also maintain a send-acknowledgement timer. When it receives any BTP packet, a peer SHALL record the packetÕs sequence number as the corresponding BTP sessionÕs pending acknowledgement value and start the send-acknowledgement timer if it is not already running. The timerÕs duration timer SHALL be defined as any value less than one-half the acknowledgement timeout interval. This ensures that on a healthy BLE connection, a peer will always receive acknowledgements for sent packets before its acknowledgement-received timer expires.

A peer SHALL stop its send-acknowledgement timer when any pending acknowledgement is sent, either as a stand-alone BTP packet or piggybacked onto an outgoing buffer segment. If this timer expires and the peer has a pending acknowledgement, the peer SHALL immediately send that acknowledgement. If the peer sends any packet before this timer expires, it SHALL piggyback any pending acknowledgement on the transmitted packet and stop the send-acknowledgement timer.

Because the serverÕs handshake response bears an implicit BTP sequence number of zero, a client SHALL set its pending acknowledgement value to zero and start its send-acknowledgement timer when it receives the serverÕs a handshake response. Operation of the send-acknowledgement and acknowledgement-received timers is illustrated in Figure 26, ÒBTP session lifecycle for Central act! ing as GATT ClientÓ in Section 4.17.3.11, ÒProtocol State DiagramsÓ.

If a peer detects that its receive window has shrunk to two or fewer free slots, it SHALL immedi! ately send any pending acknowledgement as a stand-alone BTP packet. This prevents the session from stalling in the interval between when a peerÕs receive window becomes empty and when its send-acknowledgement timer would normally fire.

4.17.3.9. Idle Connection State

When neither side of a BTP session has data to send, BTP packets will still be exchanged every send-acknowledgement interval due to acknowledgements generated by the receipt of previous data or stand-alone acknowledgement packets, as discussed in Section 4.17.3.8, ÒPacket Acknowledge! mentsÓ. The behavior of the acknowledgement-received timer in this scenario doubles as a keep-alive mechanism, as it will cause a peer to close a BLE connection automatically if the remote BTP stack crashes or becomes unresponsive. This scenario is illustrated in Figure 25, ÒIdle connection scenarioÓ.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

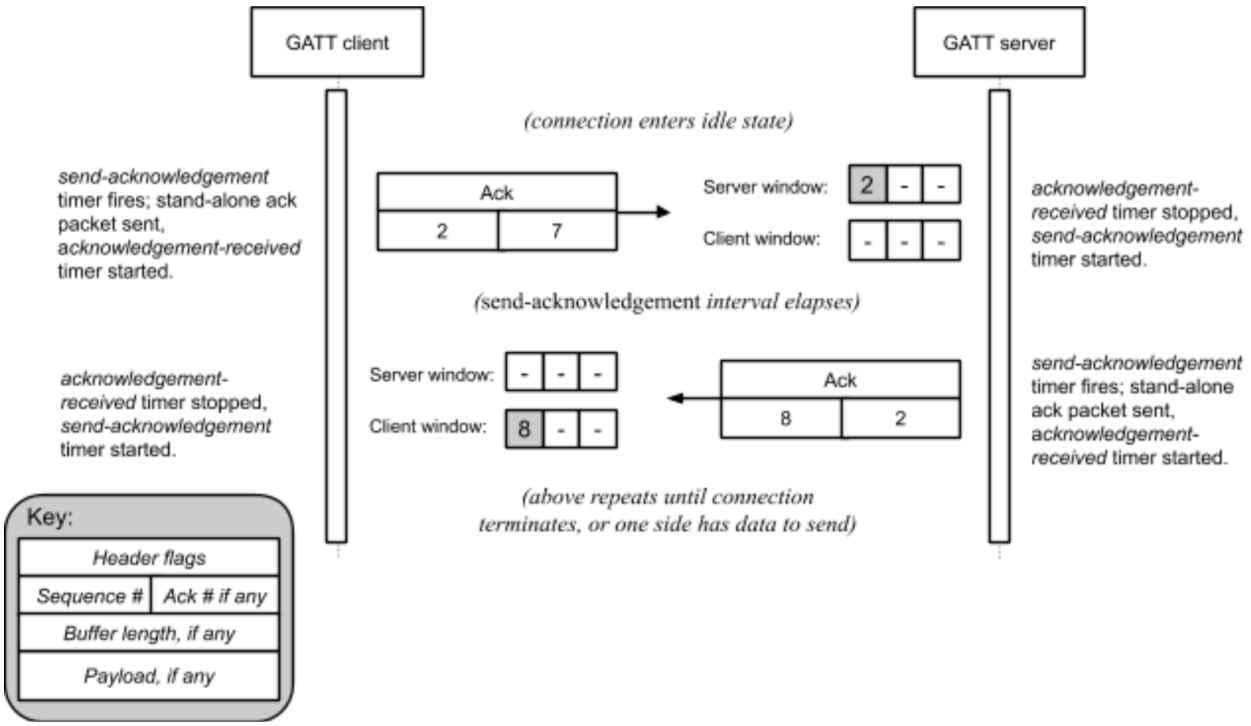


Figure 25. Idle connection scenario

4.17.3.10. Connection Shutdown

To close a BTP session, a GATT client SHALL unsubscribe from characteristic C2. The GATT server SHALL take this action to indicate closure of any BTP session open to the client.

If a BTP Server needs to close the BTP session, it SHALL terminate its BLE connection to the client.

4.17.3.11. Protocol State Diagrams

Figure 26, ÒBTP session lifecycle for Central acting as GATT ClientÓ shows the state machine for BTP session management of a BTP Client Device.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

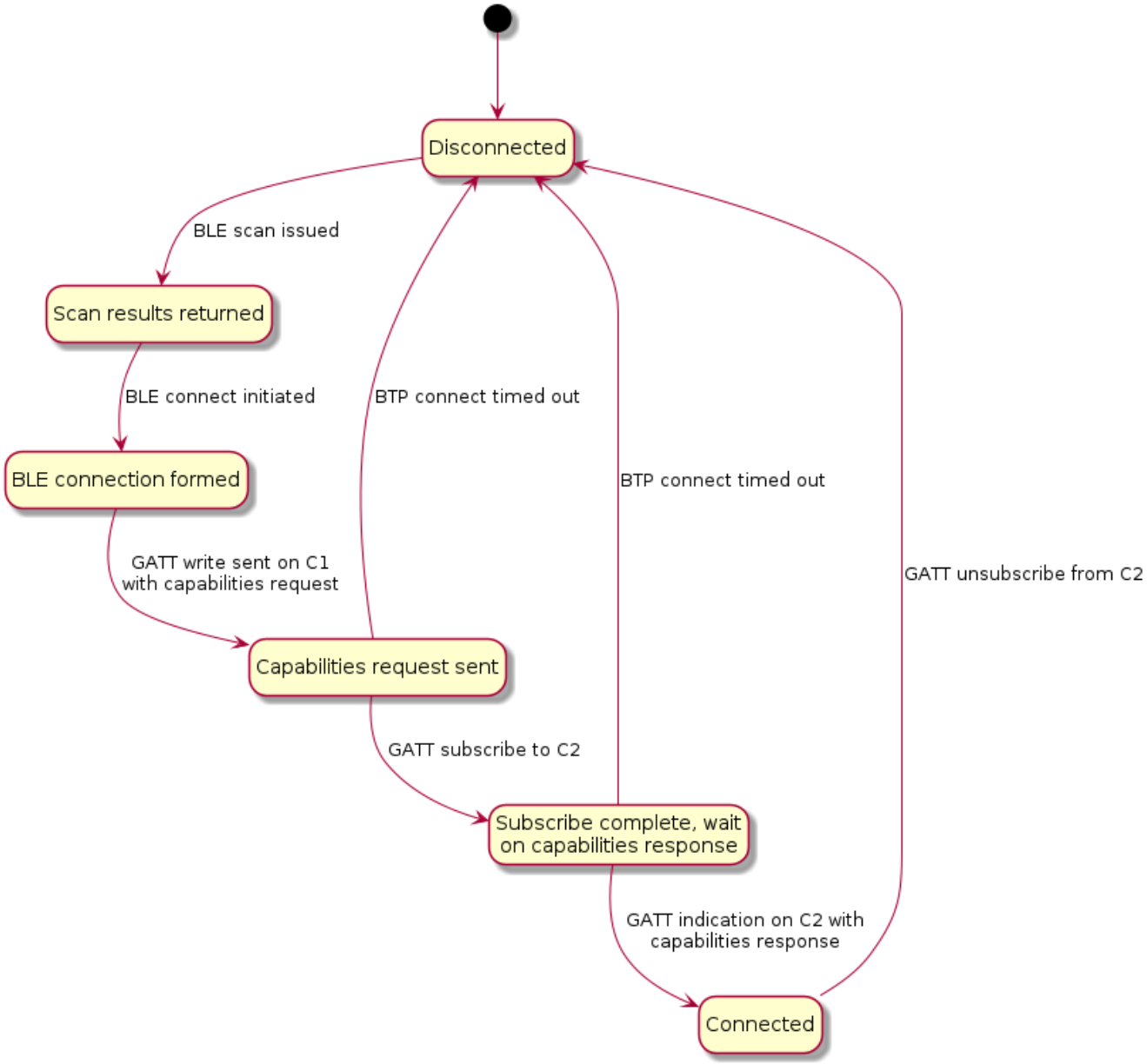


Figure 26. BTP session lifecycle for Central acting as GATT Client

Figure 27, ÒBTP session lifecycle for Peripheral acting as GATT ServerÓ shows the state machine for BTP session management of a BTP Server Device.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

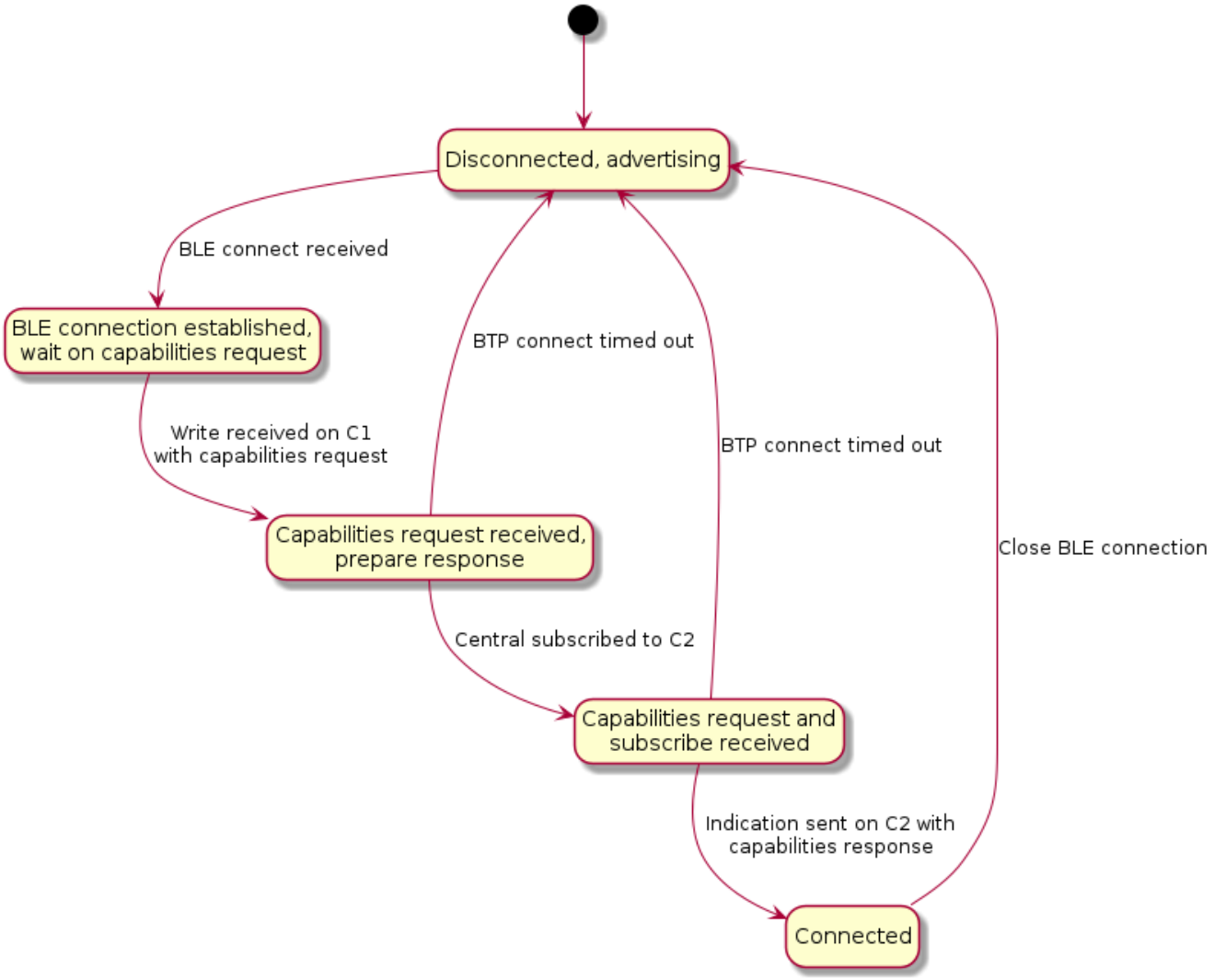


Figure 27. BTP session lifecycle for Peripheral acting as GATT Server

Note that in Figure 27, ÒBTP session lifecycle for Peripheral acting as GATT ServerÓ , the state machine is identical for GATT clients and servers with the distinction that clients send data to servers via confirmed writes, and servers send data to clients via indications.

Figure 28, ÒState diagram for BTP session post-establishmentÓ shows the state machine for BTP ses! sion maintenance at the protocol level, including liveliness enforcement through keep alive mes! sages and automatic teardown if acknowledgements are received before the timeout.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

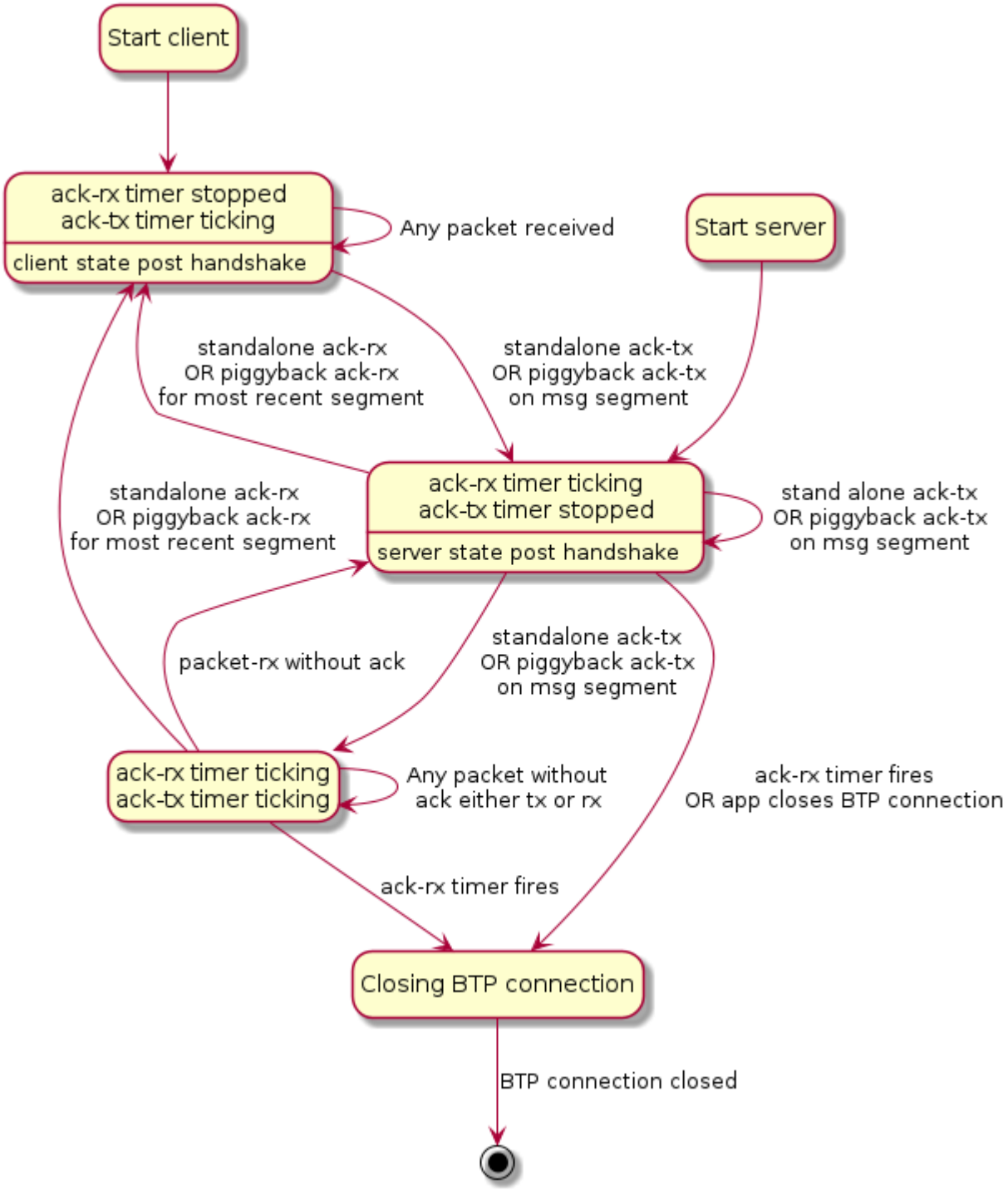


Figure 28. State diagram for BTP session post-establishment

4.17.4. Parameters and Constants

Table 32, ÒGlossary of constantsÓ is a glossary of constants used in this chapter, along with a brief description and the default for each constant.

Table 32. Glossary of constants

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Constant Name	Description	Default
BTP_CONN_RSP_TIMEOUT	The maximum amount of time after sending a BTP Session Handshake request to wait for a BTP Session Handshake response before closing the con! nection.	5 seconds
BTP_ACK_TIMEOUT	The maximum amount of time after receipt of a segment before a stand-alone ACK must be sent.	15 seconds
BTP_CONN_IDLE_TIMEOUT	The maximum amount of time no unique data has been sent over a BTP session before the Central Device must close the BTP session.	30 seconds

4.17.5. Bluetooth SIG Considerations

The UUID is provided by Bluetooth SIG, Inc. and may only be used by its members in compliance with all terms and conditions of use issued by the Bluetooth SIG, Inc. For more information, visit https://www.bluetooth.com/specifications/assigned-numbers/16-bit-uuids-for-sdos .

Use of the Bluetooth extensions feature of this specification and specifically the MATTER_BLE_SER! VICE_UUID is strictly prohibited unless the product is certified by both the Bluetooth SIG and the Connectivity Standards Alliance by a member of good standing of both organizations.

Table 33. SIG UUID assignment

Constant Name	Description	Value
MATTER_BLE_SERVICE_UUID	The UUID for the Matter-over-BLE service as assigned by the Bluetooth SIG.	0xFFF6

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Chapter 5. Commissioning

5.1. Onboarding Payload

The purpose of this section is to define the contents of the Onboarding Payload needed to allow onboarding a device into a Matter network. It also specifies the representation and encoding of said payload as a QR Code, as a manually entered code, and as content in an NFC tag.

5.1.1. Onboarding Payload Contents

The Onboarding Payload is composed of required and optional information which will be used by the Commissioner to ensure interoperability between commissioners and devices and provide a consistent user experience. Some or all of this content will be encoded into different formats, some human-readable (such as numeric string) and machine-readable (such as QR code and NFC) for! mats for printing or display on or integration into the device. The following are the elements that may be used in an Onboarding Payload for a Matter device.

5.1.1.1. Version

A version indication provides versioning of the payload and SHALL be included. Version for machine-readable formats is 3 bits with an initial version of 0b000 Version for Manual Pairing Code is 1 bit with an initial version of 0b0.
Rationale : This allows a way to introduce changes to the payload as needed going into the future.

5.1.1.2. Vendor ID and Product ID

Vendor ID and Product ID , each a 16-bit value, SHALL be included in machine-readable formats and MAY be included in the Manual Pairing Code.
Rationale : This allows a way to identify the make and model of the device, which is used further during the commissioning flow, such as during the Device Attestation procedure . These unique identifiers also help to retrieve device model metadata like product name, product description, and firmware update URL from the Distributed Compliance Ledger , as well as information relevant to the commissioning flow (see Section 5.7, ÒDevice Commissioning FlowsÓ).

5.1.1.3. Custom Flow

A 2-bit unsigned enumeration specifying the Device Commissioning Flow SHALL be included in machine-readable formats. For the encoding of Custom Flow in the Manual Pairing Code, see Sec! tion 5.1.4.1.2, ÒCustom Flow for Manual Pairing CodeÓ .
Rationale : This guides the Commissioner as to whether steps are needed before commissioning can take place.

- A value of 0 indicates that no steps are needed (apart from powering the device).
- A value of 1 indicates that user interaction with the device (pressing a button, for example) is required before commissioning can take place. The specific steps required can be found in the CommissioningModeInitialStepsHint field of the Distributed Compliance Ledger for the given Vendor ID and Product ID.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

- A value of 2 indicates that an interaction with a service provided by the manufacturer is required for initial device setup before it is available for commissioning by any Commissioner. The URL for this service can be found in the CommissioningCustomFlowUrl field of the Distributed Compliance Ledger for the given Vendor ID and Product ID.

5.1.1.4. Discovery Capabilities Bitmask

An 8-bit capabilities bitmask SHALL be included in machine-readable formats.
Rationale: The Discovery Capabilities Bitmask contains information about the deviceÕs available technologies for device discovery (see Section 5.4, ÒDevice DiscoveryÓ).

5.1.1.5. Discriminator value

A Discriminator SHALL be included as a 12-bit unsigned integer, which SHALL match the value which a device advertises during commissioning. To easily distinguish between advertising devices, this value SHOULD be different for each individual device.
For machine-readable formats, the full 12-bit Discriminator is used. For the Manual Pairing Code, only the upper 4 bits out of the 12-bit Discriminator are used.
Rationale: The Discriminator value helps to further identify potential devices during the setup process and helps to improve the speed and robustness of the setup experience for the user.

5.1.1.6. Passcode

A Passcode SHALL be included as a 27-bit unsigned integer, which serves as proof of possession during commissioning. The 27-bit unsigned integer encodes an 8-digit decimal numeric value, and therefore shall be restricted to the values 0x0000001 to 0x5F5E0FE (00000001 to 99999998 in decimal), excluding the invalid Passcode values.

Rationale: The Passcode establishes proof of possession and is also used as the shared secret in setting up the initial secure channel over which further onboarding steps take place.

5.1.1.7. TLV Data

Variable-length TLV data using the TLV format MAY be included in machine-readable formats providing optional information. More details about the TLV can be found in Section 5.1.5, ÒTLV ContentÓ.

5.1.2. Onboarding Material Representation

In order for the users of Matter products to recognize the onboarding material, and be able to use it easily, it is important to keep the representations of the onboarding material unified and of certain minimum size. To support this the Matter Brand Guidelines specify the characteristics like composition, colors, font, font size, QR Code size and digit-grouping of the Manual Pairing code.

When the onboarding material is printed on product or packaging material it SHALL follow the Matter Brand Guidelines.

Other representations (product display, app, etc) of the onboarding material SHOULD follow the Matter Brand Guidelines.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

5.1.3. QR Code

The Onboarding Payload MAY be included on (or with) a device in the form of a QR code. The fol! lowing sections detail the content, encoding, and formatting of the QR code.

5.1.3.1. Payload

The content of the QR code consists of the concatenation of a prefix string and a Base-38-encoded string containing the required and optional TLV content:

```
QR code string := <prefix><base-38-content>
```

Prefix String

The prefix string consists of the three-character string:

```
MT:
```

Base-38 Content

The required content of the QR code is composed of a Packed Binary Data Structure containing ele! ments of the Onboarding Payload as detailed below. The resulting data is Base-38 encoded (with a specific alphabet) to form a string compatible with alphanumeric QR encoding.

Packed Binary Data Structure

Individual data elements SHALL be placed into the structure in the order detailed in the table below. Elements being packed are not necessarily byte- or word-aligned. The resulting packed struc! ture is presented to the encoder as a multi-byte array (see Encoding section below), which SHALL be padded with '0' bits at the end of the structure to the nearest byte boundary.

The bits of each fixed-size value are placed in the packed binary data structure in order from least significant to most significant. If TLV Data is included, it is appended to the end of the packed binary data.

For example, the first elements in the table below SHALL be packed into the first bytes of the data array as pictured:

Table 34. Packing of the onboarding payload

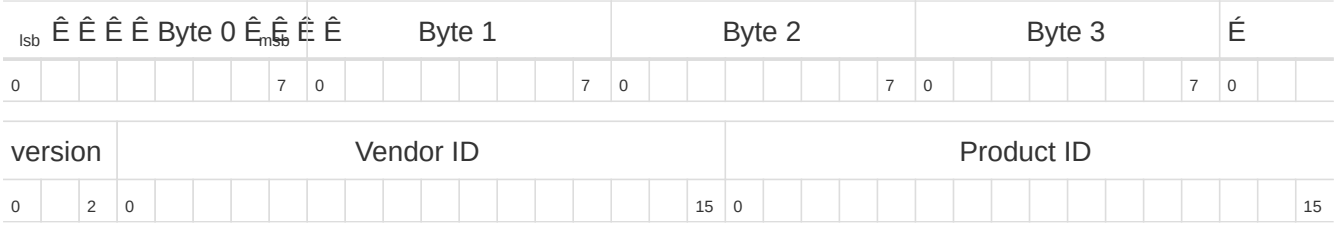


Table 35. Packed Binary Data Structure for Onboarding Payload

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Onboarding Payload Ele! ment	Size (bits)	Require d	Notes
Version	3	Yes	3-bit value specifying the QR code payload version. SHALL be 000.
Vendor ID	16	Yes	
Product ID	16	Yes	
Custom Flow	2	Yes	Device Commissioning Flow Ê 0:Standard commissioning flow : such a device, when uncom! missioned, always enters commissioning mode upon power-up, subject to the rules in Section 5.4.2.2, ÒAnnouncement Com! mencementÓ. Ê 1:User-intent commissioning flow : user action required to enter commissioning mode. Ê 2:Custom commissioning flow : interaction with a vendor-specified means is needed before commissioning. Ê 3: Reserved
Discovery Capabilities Bitmask	8	Yes	Defined in table below.
Discriminator	12	Yes	12-bit as defined in Discriminator
Passcode	27	Yes	See Section 5.1.7, ÒGeneration of the PasscodeÓ
Padding	4	Yes	Bit-padding of '0Õs to expand to the nearest byte boundary, thus byte-aligning any TLV Data that follows.
TLV Data	Variable	No	Variable length TLV data. Zero length if TLV is not included. This data is byte-aligned. See TLV Data sections below for detail.

Table 36. Discovery Capabilities Bitmask

Bit	Size (bits)	Description
0 (lsb)	1	Soft-AP: Ê 0: Device does not support hosting a Soft-AP or is currently commissioned into one or more fabrics. Ê 1: Device supports hosting a Soft-AP when not commissioned.
1	1	BLE: Ê 0: Device does not support BLE for discovery or is currently commissioned into one or more fabrics. Ê 1: Device supports BLE for discovery when not commissioned.
2	1	On IP network: Ê 1: Device is already on the IP network
3..7	5	Reserved (SHALL be 0)

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

TLV Data

The TLV data is an optional, variable-length payload. The payload is composed of one or more TLV-encoded elements as defined in detail below in the TLV Content section.

Encoding

The Packed Binary Data Structure is Base-38 encoded (with a specific alphabet) to produce an alphanumeric string suitable for use as a QR code payload.

Alphabet

The Base-38 alphabet to be employed is composed of a subset of the 45 available characters (A-Z0-9$%*+./ :-) in the QR code for alphanumeric encoding as defined by ISO/IECÊ18004:2015, with characters $, %, *, +, / , <space>, and : removed.

Table 37. Alphabet for Onboard Payload Encoding

Code	Charac!ter	Code	Charac!ter	Code	Charac!ter	Code	Charac!ter	Code	Charac!ter
00	0	09	9	18	I	27	R	36	-
01	1	10	A	19	J	28	S	37	.
02	2	11	B	20	K	29	T		
03	3	12	C	21	L	30	U		
04	4	13	D	22	M	31	V		
05	5	14	E	23	N	32	W		
06	6	15	F	24	O	33	X		
07	7	16	G	25	P	34	Y		
08	8	17	H	26	Q	35	Z		

Method

Base-38 encoding is achieved by employing a simplified strategy where every 3 bytes (24 bits) of binary source data are encoded to 5 characters of the Base-38 alphabet.

Data from the Packed Binary Data Structure are encoded starting with the first byte of the struc!ture. Three-byte chunks are formed into a 24-bit unsigned integer for encoding as follows:

$$UINT_{24} = (BYTE_{N+2} << 16) | (BYTE_{N+1} << 8) | (BYTE_{N} << 0)$$

The 24-bit value is subsequently converted to Base-38 radix using the alphabet above to produce a 5-character substring, with the least-significant character appearing first (little-endian).

If a single byte of binary source data remains, it shall be converted to Base-38 radix using the alpha!bet above to produce a 2-character substring, with the least-significant character appearing first.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

If two bytes of binary source data remains, they shall be formed into a 16-bit unsigned integer for encoding as follows:

$$UINT_{16} = (BYTE_{N+1} << 8) | (BYTE_{N} << 0)$$

This 16-bit value is subsequently converted to Base-38 radix using the alphabet above to produce a 4-character substring, with the least-significant character appearing first.

The final encoded string is a result of concatenation of all substrings, with the first-encoded sub! string appearing at the beginning of the concatenated string.

5.1.3.2. QR Code Format

The format selection, which includes the QR code Version and ECC levels as well as size and color, MAY be tailored to the requirements of the manufacturer and their respective product, provided it meets the following requirements:

QR code Version and Encoding

The QR code generated, as defined in ISO/IECÊ18004:2015, SHALL be of Version 1 or higher, using alphanumeric encoding. The size of the payload implies a minimum Version, though a higher Ver! sion may be needed to allow a higher ECC level. For example, a minimum payload of 22 alphanu! meric characters (19 base-38-encoded characters from the packed binary structure plus 3 prefix characters) can be fit into a Version 1 with ECC=L, but for ECC=M, Q or H, the same payload requires a Version 2 QR code. This allows the Manufacturer to balance between ECC, pixel size and overall size.

Example QR Code Sizes and Payloads

QR Code Version	Module Size	ECC Level	Alphanumeric capacity (chars)	Total available payload, exclud! ing prefix (bits)	Available pay! load for TLV data (bits)
1	21x21	L	25	104	16
2	25x25	L	47	208	120
		M	38	168	80
		Q	29	120	32
3	29x29	L	77	352	264
		M	61	272	184
		Q	47	208	120
		H	35	152	64

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

QR Code Version	Module Size	ECC Level	Alphanumeric capacity (chars)	Total available payload, exclud! ing prefix (bits)	Available pay! load for TLV data (bits)
4	33x33	L	114	528	440
		M	90	416	328
		Q	67	304	216
		H	50	224	136
5	37x37	L	154	720	632
		M	122	568	480
		Q	87	400	312
		H	64	288	200

NOTE Version 1 codes with ECC levels M, Q, and H and version 2 codes with ECC level H have insufficient capacity

NOTE Total available payload, excluding prefix = (trunc((N-3) / 5) * 24) where N is the number of alphanumeric characters which fit in the QR code. This formula uses N-3 to account for the prefix characters, and then determines how many groups of 5 base-38-encoded characters can fit; each such group carrying 24 bits of payload.
This formula fills groups of 5 characters after the MT: prefix. If there are 2,3 or 4 characters left after these groups, an additional 8 bits (for 2,3 characters) or 16 bits (for 4 characters) of TLV data can be accommodated. So the entries in the table take this into account.
Available payload for TLV data = (Total available payload, excluding prefix - 88) since the minimum payload for the Packed Binary Data Structure is 84 bits before padding, or 88 bits with padding.

ECC Level

The QR code SHOULD employ level M or higher ECC.

NOTE A higher level ECC does not help against typical 'reading' issues like shiny surfaces, bad contrast or issues with camera resolution/focus, and lack of camera-app processing dedicated for QR codes. Therefore, in certain situations ECC=L MAY be used as well (e.g. to prevent having to move to a higher Version to fit the payload).

5.1.4. Manual Pairing Code

This section describes the content and format of the Manual Pairing Code, which can be used in cer! tain situations next to or instead of the QR code described above.

5.1.4.1. Content

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Payload

The payload of the Manual Pairing Code consists of the following required and optional data ele! ments.

Table 38. Manual Pairing Code Elements

Element	Size (bits)	Require d	Notes
VERSION	1	Yes	Shall be 0 Version is encoded as part of first digit of the Manual Pair! ing Code. A value of 1 is reserved for future extension of the specification.
VID_PID_PRESENT	1	Yes	0: no Vendor ID and Product ID present in Manual Pairing Code 1: Vendor ID and Product ID data included
DISCRIMINATOR	4	Yes	4 Most-significant bits of the 12-bits Discriminator described above
PASSCODE	27	Yes	Same as 27-bit Passcode described above
VENDOR_ID	16	No	Needed only to support devices that need a user-intent or vendor specific flow before commissioning (i.e. a non-zero Custom Flow value). If an accompanying QR code is present on the device with the Custom Flow field set to a non-zero value, or if the device requires Custom commissioning flow , this element SHALL be included.
PRODUCT_ID	16	No*	* This element SHALL be included if and only if the VEN! DOR_ID element is present.

The Vendor ID and Product ID elements are optional. Including these may provide additional infor! mation for the setup flow at the expense of a substantially longer Manual Pairing Code.

Custom Flow for Manual Pairing Code

The encoding for Manual Pairing Code does not have a dedicated field for Custom Flow, as exists in the Packed Binary Data Structure . Instead, this information is encoded in the following way:

- For Standard commissioning flow , the variant of Manual Pairing Code without Vendor ID and Product ID SHALL be used. A commissioner encountering such Manual Pairing Code SHALL assume it is a "standard flow" device.
- For User-intent commissioning flow and Custom Commissioning flow , the variant of Manual Pairing Code with Vendor ID and Product ID SHALL be used. For this case, a commissioner SHOULD use Vendor ID and Product ID to lookup the CommissioningCustomFlow field in the Distributed Compliance Ledger to determine which of these values applies for this Vendor ID and Product ID combination.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Encoding

The required and optional elements, along with a check digit, are encoded into either an 11-digit or 21-digit decimal numeric string, depending on whether the optional Vendor and Product ID infor! mation is included.

Method

Each group of digits in the Pairing Code SHALL be encoded as described in the table below. The left-most digit of the entire string SHALL be represented by DIGIT[1] . Groups of multiple digits SHALL be encoded such that the most-significant digit appears first (left-most).

Table 39. Encoding Method without Vendor and Product IDÕs (VID_PID_Present == 0)

Digit	Contents	Encoding	Notes
1 (left-most)	- Version 0 - VID_PID present flag - 2 ms-bits of discrimina! tor	DIGIT[1] := Ê Ê (VID_PID_PRESENT << 2) | Ê Ê (DISCRIMINATOR >> 10)	Allows first digit typed/spo! ken to determine version and VID/PID present. Yields a decimal number from 0..7 (0..3 if VID,PID not present). First digit of '8' or '9' would be invalid for v1 and would indicate new format (e.g. version 2)
2..6	- 3rd and 4th ms-bits of Discriminator - 14 ls-bits of PASSCODE	DIGIT[2..6] := Ê Ê ((DISCRIMINATOR & 0x300) << 6) | Ê Ê (PASSCODE & 0x3FFF)	Yields a 5-digit decimal num! ber from 00000 to 65535 (0xFFFF/16 bits)
7..10	- 13 ms-bits of PASSCODE	DIGIT[7..10] := Ê Ê (PASSCODE >> 14)	Yields a 4-digit decimal num! ber from 0000 to 8191 (0x1FFF/13 bits)
11	- Check Digit	DIGIT[11] := Ê Ê (CHECK_DIGIT)	See Check Digit section for encoding

Table 40. Encoding Method with Vendor and Product IDÕs included (VID_PID_Present == 1)

Digit	Contents	Encoding	Notes
1 (left-most)	- Version 0 - VID_PID present flag - 2 ms-bits of Discrimina! tor	DIGIT[1] := Ê Ê (VID_PID_PRESENT << 2) | Ê Ê (DISCRIMINATOR >> 10)	Allows first digit typed/spo! ken to determine version and VID/PID present. Yields a decimal number from 0..7 (4..7 if VID,PID present). First digit of '8' or '9' would be invalid for v1 and would indicate new format (e.g. version 2)

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Digit	Contents	Encoding	Notes
2..6	- 3rd and 4th ms-bits of Discriminator - 14 ls-bits of PASSCODE	DIGIT[2..6] := Ê Ê ((DISCRIMINATOR & 0x300) << 6) | Ê Ê (PASSCODE & 0x3FFF)	Yields a 5-digit decimal number from 00000 to 65535 (0xFFFF/16 bits)
7..10	- 13 ms-bits of PASSCODE	DIGIT[7..10] := Ê Ê (PASSCODE >> 14)	Yields a 4-digit decimal number from 0000 to 8191 (0x1FFF/13 bits)
11..15	- Vendor ID	DIGIT[11..15] := Ê Ê (VENDOR_ID)	Yields a 5-digit decimal number from 00000 to 65535 (0xFFFF/16 bits)
16..20	- Product ID	DIGIT[16..20] := Ê Ê (PRODUCT_ID)	Yields a 5-digit decimal number from 00000 to 65535 (0xFFFF/16 bits)
21	- Check Digit	DIGIT[21] := Ê Ê (CHECK_DIGIT)	See Check Digit section for encoding

Check Digit

The CHECK_DIGIT element is a single decimal digit computed across all of the preceding digits of the Pairing Code using the Verhoeff algorithm.

5.1.4.2. Copying between applications

When the Manual Pairing Code is presented in an application within a multi-function device, such as an application on a smartphone, it SHOULD provide a mechanism such as a copy button to allow easy conveyance of the information to other commissioners on the same device. When a Commissioner is implemented as an application within a multi-function device, such as an application on a smartphone, it SHOULD provide a mechanism such as a paste button to allow easy conveyance of the information from an administrator on the same device.

5.1.5. TLV Content

A variable-length TLV Data section MAY be encoded into the Packed Binary Data Structure. The TLV section MAY consist of manufacturer-specific information elements and/or elements common to Matter, encoded using TLV. All elements SHALL be housed within an anonymous top-level structure container.

5.1.5.1. Manufacturer-specific Elements

Manufacturer-specific elements SHALL be tagged with context-specific tags that have semantics which are defined by the vendor for use in the products using their Vendor ID, and SHALL use tag numbers 0x80 to 0xFF.
Tag numbers 0x00 to 0x7F are reserved to indicate Matter-common elements.

Manufacturer-specific elements inherit the context of the Vendor ID and Product ID provided in the Packed Binary Data Structure described above. All elements SHALL follow the constraints outlined

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

in Appendix A, Tag-length-value (TLV) Encoding Format .

5.1.5.2. Matter-common Elements

All elements common to Matter SHALL use tag numbers in the range 0x00 to 0x7F, as defined in the following section.

Vendors are encouraged to use Matter-common elements where applicable.

Table 41. Matter-common Reserved Tags

Tag	Value	Description	Type(s)
kTag_SerialNumber	0x00	Device Serial #	UTF-8 String (length = 1..32 bytes)
			Unsigned Integer, up to 8-byte value (has room to represent a 19-digit decimal number)
PBKDFIterations *	0x01	PBKDFParameterSet Iterations	Unsigned Integer (range = CRYPTO_PBKDF_ITERATIONS_MIN.. CRYPTO_PBKDF_ITERATIONS_MAX)
PBKDFSalt *	0x02	PBKDFParameterSet Salt	Octet String (length = 16..32 bytes)
kTag_NumberOfDevices	0x03	Number of devices that are expected to be onboarded using this payload when using the Enhanced Commissioning Method	Unsigned Integer, range 1 to 255
kTag_CommissioningTimeout	0x04	Time, in seconds, during which the device(s) are expected to be commissionable using the Enhanced Commissioning Method	Unsigned Integer, see Announcement Duration
reserved	0x05..0x7F	reserved for future use	

* If the PBKDF parameters are to be included in the TLV section, both the PBKDFSalt and PBKDFIterations SHALL be encoded.

5.1.5.3. TLV Examples

Manufacturer-specific and Matter-common elements

```
Ê {
Ê    vendorTag01 (0x81) = "Vendor",
Ê    kTag_SerialNumber(0) = "1234567890"
Ê }
```

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

The above notation encodes to the following bytes:

```
0x15 0x2C 0x81 0x06 0x56 0x65 0x6E 0x64 0x6F 0x72 0x2C 0x00 0x0A 0x31 0x32 0x33
0x34 0x35 0x36 0x37 0x38 0x39 0x30 0x18


Data       Comments
========== ================================================
0x15       Control Byte for outermost container (structure)
Ê          - Tag control 000xxxxxb: Anonymous tag
Ê          - Elem type   xxx10101b: Structure

0x2C       Control Byte for next TLV
Ê          - Tag control 001xxxxxb: Context-specific tag
Ê          - Elem type   xxx01100b: UTF-8 String, 1-byte length
0x81       Context-specific vendor tag
Ê          - Matter-common versus vendor tag 1xxxxxxxb: Vendor tag
Ê          - Tag number                 x0000001b: Vendor tag #1
Ê                                     ---------
Ê                                     10000001b = 0x81
0x06       Length of vendor string (e.g. 6 bytes)
0x56 0x65 0x6E 0x64 0x6F 0x72
Ê          UTF-8 encoded vendor string "Vendor"

0x2C       Control byte for next TLV
Ê          - Tag control 001xxxxxb: Context-specific tag
Ê          - Elem type   xxx01100b: UTF-8 String, 1-byte length
Ê                     ---------
Ê                     00101100b = 0x2C
0x00       Context-specific Matter-common Serial Number tag
Ê          - Matter-common versus vendor tag 0xxxxxxxb: Matter-common tag
Ê          - Tag number                 x0000000b: kTag_SerialNumber
Ê                                     ---------
Ê                                     00000000b = 0x00
0x0A       Length of Serial Number string (10 bytes)
0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x30
Ê          UTF-8 encoded Serial Number string "1234567890"

0x18       End of container
```

Matter-common elements only

```
Ê  {
Ê     kTag_SerialNumber (0) = "1234567890"
Ê  }
```

The above notation encodes to the following bytes:

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

```
0x15 0x2C 0x00 0x0A 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x30 0x18

Data       Comments
========== ================================================
0x15       Control Byte for outermost container (structure)
Ê          - Tag control 000xxxxxb: Anonymous tag
Ê          - Elem type   xxx10101b: Structure

0x2C       Control Byte for next TLV
Ê          - Tag control 001xxxxxb: Context-specific tag
Ê          - Elem type   xxx01100b: UTF-8 String, 1-byte length
Ê                        ---------
Ê                        00101100b = 0x2C
0x00       Context-specific Matter-common Serial Number tag
Ê          - Matter-common versus vendor tag 0xxxxxxxb: Matter-common tag
Ê          - Tag number                    x0000000b: kTag_SerialNumber
Ê                                    ---------
Ê                                    00000000b = 0x00
0x0A       Length of Serial Number string (10 bytes)
0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x30
Ê          UTF-8 encoded Serial Number string "1234567890"

0x18       End of container
```

5.1.6. Concatenation

The Onboarding Payload MAY be concatenated with additional Onboarding Payloads to be placed in a single QR Code:

```
Ê  QR code string := MT:<onboarding-base-38-content>*<onboarding-base-38-content2>
```

Where * is used as the delimiter.

Concatenation of multiple Matter Onboarding Payloads allows a single QR code to provide the onboarding payload for a number of devices. Example use case for this concatenation:

¥ Easy onboarding for multi-device packaging, e.g. for a package of light bulbs containing four separate bulbs. Each bulb will have its own Onboarding Payload code(s) printed on the bulb itself. The Manufacturer MAY include a leaflet in the box with a larger QR code containing the concatenation of the four individual Onboarding Payloads. The user can then scan this com! bined QR code (one step for the user) which would give the Commissioner the Onboarding Pay! load for all four bulbs in one operation, and it can proceed to commission the four bulbs.

All Commissioners SHALL recognize the * separator from the QR code as indication concatenation is used.
A Commissioner which does not support such concatenated Matter Onboarding Payloads SHOULD indicate to the user the need to commission devices one by one by scanning their individual QR codes.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

The Commissioner SHOULD commission the devices in the order as they are provided in the concatenated code. (This ordering is particularly relevant in case of combi-packs where one of the devices needs to be commissioned first, e.g. a Thread Router first, then one or more Thread-connected bulbs).

Example of concatenated Onboarding Payloads:

```
MT:<onboarding-base-38-content_bulb1>*<onboarding-base-38-content_bulb2>*<onboarding-base-38-content_bulb3>*<onboarding-base-38-content_bulb4>
```

5.1.7. Generation of the Passcode

A device can support either dynamic or static passcodes for purposes of establishing the shared secret for the initial secure channel over which further onboarding steps take place.

All devices SHALL conform to the following rules for passcodes:

- Passcodes SHALL NOT be derived from public information, such as a serial number, manufacturer date, MAC address, region of origin, etc.
- The Passcode generation process SHALL use a cryptographically secure random number generator.

If a device generates a dynamic Passcode, then it SHALL conform to the following additional rule:

- Passcodes SHALL be accessible to commissioner only during the commissioning process.

If a device cannot generate a dynamic Passcode, then the static Passcode SHALL conform to the following additional rules:

- A random passcode SHALL be generated and used for each individual device.
- The device SHALL be supplied with the PAKE verifier in its internal storage.
- If the static passcode is also supplied to the device, the static passcode SHALL NOT be accessible during operational mode using any data model attributes or commands.
- If the static passcode is supplied to the device, its storage location SHALL be physically isolated from the location where the PAKE verifier is stored and SHALL only be accessible through local interfaces and SHALL NOT be accessible to the executing unit handling the PAKE verifier. For example, a device equipped with a NFC connected tag may contain the QR code containing the static passcode in the NFC connected tag private memory and the NDEF record containing the NFC tag onboarding payload is only presented to the commissioner during the commissioning window through the NFC interface.

5.1.7.1. Invalid Passcodes

The following Passcodes SHALL NOT be used for the PASE protocol due to their trivial, insecure nature:

- 00000000

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

- 11111111
- 22222222
- 33333333
- 44444444
- 55555555
- 66666666
- 77777777
- 88888888
- 99999999
- 12345678
- 87654321

5.1.8. NFC Tag

A Commissioner MAY use, in addition to the QR Code Format and Manual Pairing Code as described above, an NFC tag associated with a Commissionee to retrieve the Onboarding Payload. When an NFC tag is used the following requirements are applicable.

- The data contained in the NFC tag SHALL be the same as specified in QR Code Format.
- The NFC tag SHALL be one of the types as defined by NFC Forum.
- The NFC tag SHALL use the NFC Data Exchange Format (NDEF) as defined by NFCÊNDEFÊ1.0
- The NFC tag SHALL use NDEF messages as defined by NFCÊRTDÊ1.0
- The Onboarding Payload for the NFC tag SHALL use NDEF URI Record Type Definition as defined by NFCÊRTDÊURIÊ1.0 and as specified in the following table.

Table 42. NFC NDEF Representation

Offset	Content	Description
0	0xD1	TNF=0x01, SR=1, MB=1, ME=1
1	0x01	Length of Record Type
2	URI payload size in bytes	Length of payload
3	0x55	Record Name ("U")
4	0x00	URI Identifier Code: No URI abbreviation
5	URI data	MT:<base-38-content>

5.2. Initiating Commissioning

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

5.2.1. Purpose and Scope

The process of Matter commissioning can be initiated by the User in a number of ways. This section describes different user journeys supported by Matter. For each, a rationale is provided along with a high-level flow description, up until the point where a commissioning secure session is estab! lished. References to sections describing dependent functionality in more detail are provided.

The purpose of this section is to connect features provided in other sections to the user journeys for which they are designed.

WARNING	The list of user journeys provided here is not meant to be exhaustive; there may be other journeys not listed here which can be realized using Matter.

This section provides rationales for Matter functionality and does NOT contain normative require! ments for Matter.

The following User Journeys are described in this section:

- ¥ Section 5.2.2.1, ÒCommissioner Setup Code Entry, Not Yet Commissioned DeviceÓ . "Launch Com! missioner, Enter Code"
- ¥ Section 5.2.2.2, ÒUser-Initiated Beacon Detection, Not Yet Commissioned DeviceÓ . "Launch Com! missioner, Discover New Devices"
- ¥ Section 5.2.2.3, ÒUser-Initiated Beacon Detection, Already Commissioned DeviceÓ . "Launch Com! missioner, Discover My devices"
- ¥ Section 5.2.2.4, ÒCommissioner Discovery, from an On-Network DeviceÓ . "Launch Device User Interface, Discover Commissioners"

5.2.2. User Journey Details

5.2.2.1. Commissioner Setup Code Entry, Not Yet Commissioned Device

"Launch Commissioner, Enter Code"

In the Setup Code Entry for a Not Yet Commissioned Device use case, the User first initiates an inter! action with a Commissioner, and then provides the necessary setup code from the Commissionee, by scanning an Onboarding Payload (e.g. QR Code) or otherwise inputting the manual setup code through an input method supported by the commissioner.

5.2.2.1.1. Rationale

In this use case, the user will often have the device in-hand, have immediate access to the onboard! ing payload, and have immediate access to the desired Commissioner.

5.2.2.1.2. High Level Flow

1. User initiates an interaction with a Commissioner.
2. User inputs the onboarding payload from the Commissionee.
3. Commissioner determines which technologies to use for Device Discovery . When attempting to

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

locate the device on IP-bearing networks, the Commissionable Node Discovery method is used and typically the DNS-SD service subtypes for long or short discriminator, and commissioning mode (see Commissioning Subtypes) are specified to filter the results to Commissionees that match the discriminator in the onboarding payload and that are in Commissioning Mode. When attempting to locate the device via BLE or Soft-AP advertisements, the discriminator will typi! cally be used to filter the results.

4. Commissioner begins the Commissioning process (see Section 5.5, ÒCommissioning FlowsÓ). If more than one Commissionee is discovered, the Commissioner may further refine the results using any additional information such as a Vendor ID or Product ID that may be available in the onboarding payload. If there is still more than one discovered Commissionee, the Commissioner will typically attempt to establish a PASE secure commissioning session with each.

5.2.2.1.3. Misuse Considerations

When a device has a static onboarding payload, and the value is physically affixed to the product, it is possible for an attacker with one-time physical access to the device to obtain the onboarding pay! load and use it to compromise the security of the device in the future. For example, if the device is commissioned again using the same onboarding payload (for example, after a reset), then the attacker may be able to perform a person-in-the-middle attack which could result in a compromise of sensitive user data such as network credentials if passed to the device.

When a device includes device-specific information such as Vendor ID and Product ID in advertise! ments, then a malicious actor within advertisement range can detect this information and poten! tially associate it with the location of the device (and potentially, additional information about the location, such as its residents) in ways that the user did not intend.

5.2.2.2. User-Initiated Beacon Detection, Not Yet Commissioned Device

"Launch Commissioner, Discover New Devices"

In the User-Initiated Beacon Detection for a Not Yet Commissioned Device use case, the User first initiates an interaction with a Commissioner, and then indicates an intention to commission devices without providing additional information about them (no onboarding payload, etc).

5.2.2.2.1. Rationale

In this use case, the user has immediate access to a Commissioner. However, the user may not know how to locate the onboarding payload (it may be hidden behind a panel, pin-protected in a settings menu, or inaccessible on a device already physically installed).

Example User interactions with the Commissioner include pushing a "Discover New Devices" but! ton, or speaking to a voice agent "Agent, discover new devices".

5.2.2.2.2. High Level Flow

1. User initiates an interaction with a Commissioner.
2. User indicates an intention to commission devices without providing additional information about them.
3. Commissioner determines which technologies to use for Device Discovery . When attempting to

csa connectivity standards alliance Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

locate the device on IP-bearing networks, the Commissionable Node Discovery method is used and typically the subtype for commissioning mode (see Commissioning Subtypes) is specified with value 1 in order to filter the results to Commissionees that are in Commissioning Mode.

4. Commissioner constructs a list of Commissionees discovered, using as much information as possible from the Commissionee advertisement. When a Vendor ID and Product ID is provided in the advertisement, the Commissioner may obtain human readable descriptions of the Vendor and Product in order to assist the user with selection by using fields such as ProductName and ProductLabel from the Distributed Compliance Ledger or any other data set available to it. The ledger entry may also include additional URLs which the Commissioner can offer to the user to help in locating the Setup Code or otherwise assist in setting up the device such as the UserManualUrl , SupportUrl , and ProductUrl . The Commissioner may have additional data sets available for assisting the user.
5. User selects Commissionee from list.
6. Commissioner instructs the user to locate and input the onboarding payload.
7. Commissioner begins the Commissioning process (see Section 5.5, ÒCommissioning FlowsÓ).

5.2.2.2.3. Variation - Filter by Device Type

The user may indicate the type of device to the Commissioner when initiating this flow. For example, the user might speak the following to a voice agent: "Agent, Discover TVs".

When discovering TVs or any other specific device type on the IP network, this flow will be the same except that a subtype which specifies the device type identifier (see Descriptor Cluster on root node endpoint) is passed to the Commissionable Node Discovery method (see Commissioning Subtypes).

5.2.2.2.4. Misuse Considerations

In addition to the Misuse Considerations for the Section 5.2.2.2, ÒUser-Initiated Beacon Detection, Not Yet Commissioned DeviceÓ , a Commissioner which performs Device Discovery without knowledge of the Onboarding Payload may discover advertisements from devices that the user did not intend to onboard with the given Commissioner. This additional information collected by the Commissioner can be associated with the user in ways that the user did not intend.

5.2.2.3. User-Initiated Beacon Detection, Already Commissioned Device

"Launch Commissioner, Discover My Devices"

In the User-Initiated Beacon Detection for an Already Commissioned Device use case, the User first initiates an interaction with a Commissioner, and then indicates an intention to commission devices already on the IP network without providing additional information about them.

5.2.2.3.1. Rationale

A Device may choose to be discoverable by entities on the local IP network, even when not in Commissioning Mode, in order to satisfy specific user journeys. For example, a TV or Bridge device may choose to be discoverable in order to facilitate connectivity with other Smart Home systems.

Example User interactions with the Commissioner include pushing a "Discover My Devices" button,

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

or speaking to a voice agent "Agent, discover my devices".

5.2.2.3.2. High Level Flow

1. User initiates an interaction with a Commissioner.
2. User indicates an intention to commission existing devices on the IP network without providing additional information about them.
3. Commissioner sends the Commissionable Node Discovery broadcast message.
4. Commissioner constructs a list of Commissionees discovered, using as much information as pos! sible from the Commissionee advertisement. When a Vendor ID and Product ID is provided (see Commissioning VID/PID), the Commissioner may obtain human readable descriptions of the Vendor and Product in order to assist the user with selection by using fields such as ProductName and ProductLabel from the Distributed Compliance Ledger or any other data set available to it. The ledger entry may also include additional URLs which the Commissioner can offer to the user to help in locating the Setup Code or otherwise assist in setting up the device such as the UserManualUrl, SupportUrl , and ProductUrl . The Commissioner may have additional data sets available for assisting the user. When the Device Type (see Commissioning Device Type) and/or the Device Name (see Commissioning Device Name) values are provided, then the Commis! sioner may provide this information to the user in order to assist with Commissionee selection.
5. User selects Commissionee from list.
6. The Commissionable Node Discovery DNS-SD TXT record for the selected Commissionee includes key/value pairs that can help the Commissioner to guide the user through the next steps of the commissioning process. If the Commissioning Mode value (see Commissioning Com! missioning Mode) is set to 0, then the Commissionee is not yet in Commissioning Mode and the Commissioner can guide the user through the steps needed to put the Commissionee into Com! missioning Mode. The Pairing Hint (see: Commissioning Pairing Hint) and the Pairing Instruc! tion (see: Commissioning Pairing Instruction) fields would then indicate the steps that can be followed by the user to put the device into Commissioning Mode.
7. If not already in Commissioning Mode, Commissioner instructs the user to put the Commis! sionee into Commissioning Mode, and verifies the new state using Commissionable Node Dis! covery .
8. Commissioner instructs the user to locate and input the onboarding payload. When a Vendor ID and Product ID is available to the Commissioner, the Distributed Compliance Ledger may also provide a URL for the Device User Guide which can contain additional information to help in locating the onboarding payload. The Commissioner may have additional data sets available for assisting the user.
9. Commissioner begins the Commissioning process (see Section 5.5, ÒCommissioning FlowsÓ).

5.2.2.3.3. Variation - Filter by Device Type

The user may indicate the type of device to the Commissioner when initiating this flow. For exam! ple, the user might speak the following to a voice agent: "Agent, Discover TVs".

When discovering TVs or any other specific device type on the IP network, this flow will be the same except that a subtype which specifies the device type identifier is passed to the Commission! able Node Discovery method (see Commissioning Subtypes).

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

5.2.2.3.4. Misuse Considerations

When a Device implements Commissionable Node Discovery while not in Commissioning Mode, the time period during which it may unintentionally provide information to a malicious actor on the network is longer than it otherwise would be. This additional information could potentially be asso! ciated with the user in ways that the user did not intend. See Commissionable Node Discovery Pri! vacy Considerations for device requirements relating to this risk.

When a device includes device-specific information such as Vendor ID, Product ID and Device Type, or user-generated data such as Device Name, in the DNS-SD TXT record, then a malicious actor on the network can detect this information and potentially associate it with the user in ways that the user did not intend.

A Commissioner which performs Device Discovery without knowledge of the Onboarding Payload may discover devices on the network that the user did not intend to onboard with the given Com! missioner. This additional information collected by the Commissioner can be associated with the user in ways that the user did not intend.

5.2.2.4. Commissioner Discovery, from an On-Network Device

"Launch Device User Interface, Discover Commissioners"

In the Commissioner Discovery use case for a Device already on the IP network, the User first initi! ates an interaction with the Device via a display or other user interface, and indicates the intention to have this device commissioned by a Commissioner on the network. The Device might already have been commissioned into one or many Fabrics or it might not yet have been commissioned. Upon this user interaction, the Device discovers candidate Commissioners and allows the user to select one. The Device then requests from that Commissioner to be commissioned.

5.2.2.4.1. Rationale

In this use case, a Device (Commissionee) with a user interface, such as a TV or Thermostat, initiates the commissioning process. For example, this might be done from within a settings menu for Smart Home control. The Device discovers Commissioners on the IP-bearing network, presents the result! ing list to the User for selection. Once selected, the Device indicates to the selected Commissioner that it has been selected by the User, the Device enters Commissioning Mode and provides the onboarding payload to the User.

Another example for this use case is a Device or Node (Commissionee) with a user interface, such as a Content Provider Device or Application, that initiates the commissioning process. This might be done from a program guide or while watching a video when the user indicates a desire to play the selected content on a nearby device. The Device discovers Commissioners on the IP-bearing net! work, presents the resulting list to the User for selection. Once selected, the Commissionee indicates to the selected Commissioner that it has been selected by the User (see User Directed Commission! ing), the Commissionee enters Commissioning Mode and provides the onboarding payload to the User.

5.2.2.4.2. High Level Flow

1. User initiates an interaction with the Device.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

2. User indicates a desire to connect this Device with a Commissioner on the network.
3. Device uses Commissioner Discovery over DNS-SD on the IP bearing network.
4. Device collects candidates from DNS-SD service records found.
5. Device displays list of Commissioners discovered, including as much information as possible from the DNS-SD TXT record. When a Vendor ID and Product ID is provided (see Commissioning VID/PID), the Device may obtain human readable descriptions of the Vendor and Product in order to assist the user with selection by using fields such as ProductName and ProductLabel from the Distributed Compliance Ledger or any other data set available to it. The Device may have additional data sets available for assisting the user. When the Device Type (see Commissioning Device Type) and/or the Device Name (see Commissioning Device Name) values are provided in the DNS-SD TXT record, then the Device may provide this information to the user in order to assist with Commissioner selection.
6. User selects an entry from the list.
7. Device enters Commissioning Mode.
8. Device displays onboarding payload to the user.
9. Device initiates a User Directed Commissioning session with the selected Commissioner, which includes in the DNS-SD service name of the Device.
10. Commissioner prompts user to confirm intention to commission this device and asks for onboarding payload.
11. User enters onboarding payload into Commissioner UX.
12. Commissioner begins the commissioning process (see Section 5.5, ÒCommissioning FlowsÓ).

5.2.2.4.3. Misuse Considerations

In addition to the Misuse Considerations for the Section 5.2.2.3, ÒUser-Initiated Beacon Detection, Already Commissioned DeviceÓ , a Commissionee which performs Commissioner Discovery may dis! cover Commissioners on the network that the user did not intend to be discovered by the given Commissionee. This additional information collected by the Commissionee can be associated with the user in ways that the user did not intend. See Commissioner Discovery Privacy Considerations for Commissioner requirements relating to this risk.

Since there are no trust mechanisms employed for Commissioners advertising themselves, Commis! sionees may provide Commissioner selection choices to the User that are from malicious entities masquerading as commissioners.

When a Commissioner includes device-specific information such as Vendor ID, Product ID and Device Type, or user-generated data such as Device Name, in the DNS-SD TXT record, then a mali! cious actor on the network can detect this information and potentially associate it with the user in ways that the user did not intend.

5.3. User Directed Commissioning

csa connectivity standards alliance Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

5.3.1. Overview

In User Directed Commissioning (UDC), the Commissionee sends a message to the Commissioner in order to initiate the commissioning process (see Section 5.5, ÒCommissioning FlowsÓ).

The availability of the UDC protocol is advertised through Commissioner Discovery service records of DNS-SD service type _matterd._udp (see Section 4.3.3, ÒCommissioner DiscoveryÓ).

Overall, the UDC protocol is a lightweight "door bell" message sent by a Commissionee, and consists of an Identification Declaration which provides the CommissioneeÕs _matterc._udp DNS'SD service instance name.

Upon receiving this message, the Commissioner MAY query the DNS-SD service instance indicated in the Identification Declaration , including TXT records, in order to obtain additional information about the Commissionee, MAY obtain the corresponding Onboarding Payload from the user for this Commissionee, and MAY initiate the commissioning process with it.

One possible user journey for this feature is described in Commissioner Discovery from an Existing Device.

Figure 29. Overview of the UDC Protocol

The Commissionee is the Initiator and the Commissioner is the Recipient.

It is assumed that the user has directed the Initiator to send this message to the Recipient. Upon receipt and before starting a PASE session with the Initiator, it is assumed that the Recipient will query the DNS-SD records for the Initiator, including all TXT records, and then prompt the user for approval and to enter its Onboarding Payload.

5.3.2. UDC Protocol Messages

Table 43. User Directed Commissioning Protocol

Protocol Opcode	Protocol Command Name	Description
Protocol ID = PROTOCOL_ID_USER_DIRECTED_COMMISSIONING		
0x00	IdentificationDeclaration	The Identification Declaration message provides the DNS-SD Instance Name of the commissionee requesting com! missioning to the commissioner selected by the user.

The following defines the Matter User Directed Commissioning TLV protocol:

```
namespace matter.protocols {
Ê user-directed-commissioning => PROTOCOL [
Matter: PROTOCOL_ID_USER_DIRECTED_COMMISSIONING
```

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

```
Ê {
Ê  IdentificationDeclaration =>        IdentificationDeclaration-struct
Ê }
}
```

5.3.3. Message format

All UDC messages SHALL be structured as specified in Section 4.4, ÒMessage Frame FormatÓ.

All UDC messages are unsecured at the message layer:

- ¥ The Session ID field SHALL be set to 0.
- ¥ The Session Type bits of the Security Flags SHALL be set to 0.
- ¥ The S Flag and DSIZ fields of the Message Flags SHALL be set to 0.

The R Flag of the Exchange Flags for the UDC messages SHALL be set to 0.

For each UDC message, the application payload is the TLV encoding of the message structure as defined below:

Table 44. UDC Messages

Message Name	Payload TLV Encoding
IdentificationDeclaration	IdentificationDeclaration-struct

The other fields of the Message format are not specific to the UDC messages.

5.3.4. Message Exchanges

The flags of the Exchange Flags of the Protocol Header are defined as follows per UDC message:

Message	I Flag
IdentificationDeclaration	1

All UDC messages SHALL be sent unreliably, to an IP address found in a AAAA record associated with the Commissioner Discovery (_matterd._udp) service, using UDP with a destination port as found in the _matterd._udp SRV record. The Initiator MAY send up to 4 retries. Each retransmission SHALL be delayed by at least 100ms from the previous transmission.

The other fields of the Protocol Header are not specific to the UDC messages.

5.3.5. IdentificationDeclaration Message

This message serves to identify the commissionee. It is sent by the commissionee to the commis! sioner. The commissionee SHALL:

1. Construct the instanceNamebased upon the DNS-SD instance name defined in Commissionable Node Discovery .

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

2. Construct and send IdentificationDeclaration .

```
IdentificationDeclaration-struct => STRUCTURE [ tag-order ]
{
Ê instanceName      [1] : OCTET STRING [ length 8 ]
}
```

5.4. Device Discovery

5.4.1. Purpose and Scope

The purpose of this section is to describe the process by which a device is discovered in order to commission it onto an operational Fabric.

Depending on the networking technologies supported by a device, discovery and commissioning are possible using Bluetooth Low Energy (BLE), Wi-Fi (IEEEÊ802.11-2020) technologies, or over IP if a device is already on an IP network.

Devices that utilize Thread (IEEE 802.15.4) networking technology must also support BLE for the purpose of discovery and commissioning. Directly utilizing Thread-based commissioning for device discovery and commissioning is neither specified nor supported.

BLE commissioning utilizes the Generic Access Profile (GAP) for discovery and for connection establishment, and the Generic Attribute Profile (GATT) for credential conveyance.

Wi-Fi commissioning utilizes Soft-AP functionality where the device acts as an Access Point (AP) that doesnÕt provide Internet connectivity. Standard Wi-Fi AP advertisement and connection proto! cols are employed for device discovery and credential conveyance, respectively.

If a device already has network connectivity (over Wi-Fi, Ethernet, or otherwise) a Commissioner may discover such a device using DNS-based Service Discovery (DNS-SD), conveying credentials to the device over IP.

5.4.2. Announcement by Device

This section describes how devices announce their commissionable status to allow a Commissioner to discover the device to be commissioned.

5.4.2.1. Technology Priority

A device SHALL announce in any order of priority on all of the networking technologies it supports as indicated in the Discovery Capability Bitmask (see Table 36, ÒDiscovery Capabilities BitmaskÓ). A Commissioner that is aware of the deviceÕs Discovery Capability Bitmask SHALL initiate Device Dis! covery in any order of priority on all of the networking technologies that are supported by both the Commissioner and the device. A Commissioner that is unaware of the deviceÕs Discovery Capability Bitmask SHALL initiate Device Discovery in any order on all of the networking technologies it sup! ports out of Wi-Fi Soft-AP, BLE, and on IP network discovery.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Commissioners SHALL always support discovering a device using DNS-based Service Discovery (DNS-SD) for commissioning, irrespective of the Discovery Capabilities Bitmask specified in the Section 5.1.1, ÒOnboarding Payload ContentsÓ .

5.4.2.2. Announcement Commencement

A device which is not yet commissioned into a Matter fabric SHALL commence announcing its abil! ity to be commissioned depending on its primary device function and manufacturer-chosen Device Commissioning Flow , per the following table. Nodes already commissioned into one or more Matter fabrics and wishing to announce SHALL ONLY do so using DNS-SD over their operational network (see Section 4.3, ÒDiscoveryÓ). In the interest of privacy, an already-commissioned Node SHALL NOT commence announcement using Bluetooth LE or Wi-Fi Soft-AP technologies.

Primary Device Function	Announcement
Most control originates from a Fabric (excluding Locks and Barrier Access Devices)	SHALL start announcing automatically upon application of power when using Standard commissioning flow . When using User-intent commissioning flow or Custom Commis! sioning flow , it SHALL NOT start announcing automati! cally upon application of power.
Most control does not originate from a Fabric (e.g., dishwasher, coffee maker, refrigerator)	SHALL NOT start announcing automatically upon applica! tion of power. User-intent commissioning flow or Custom Commissioning flow is required.
Locks and Barrier Access Devices	SHALL NOT start announcing automatically upon applica! tion of power. User-intent commissioning flow or Custom Commissioning flow is required.

Note that the above guidelines are in place to avoid unnecessary pollution of the 2.4 GHz spectrum and as a mitigation of the privacy threat created due to unnecessary transmissions by a commis! sionable device.

If announcement has ceased (see Section 5.4.2.3, ÒAnnouncement DurationÓ), it may be re-initiated via a device-specific user interaction such as a button press or other action defined by the manufac! turer and indicated by the methods specified in Section 5.7, ÒDevice Commissioning FlowsÓ .

5.4.2.3. Announcement Duration

In order to minimize unnecessary pollution of the crowded 2.4 GHz wireless spectrum, a commis! sionable device SHALL NOT announce for a duration longer than 15 minutes after announcement commences. This should provide ample time for a user to commission a range of devices, including time to download, install and launch applications, transit rooms within a home, etc.

Note that devices MAY choose to announce for less time in order to conserve battery life or for other device-specific reasons. Note that an announcement duration that is too short may result in a poor setup experience for users. Shorter announcement intervals SHOULD only be employed to meet otherwise unattainable device functionality/requirements. To help strike a balance between a good setup experience and conserving battery life, a device SHALL NOT announce for a duration of less than 3 minutes after announcement commences.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

A failed attempt to commission does not restart or delay the timeout. Moreover, this timeout applies only to cessation of announcements and not to abortion of connections, i.e., a connection SHOULD NOT abort prematurely upon expiration of the announcement duration.

5.4.2.4. Discovery Information

This section details the information advertised by a commissionable Node.

Field	Length	Is Required?
Discriminator	12-bit	Yes
Vendor ID	16-bit	No
Product ID	16-bit	No
Extended Data	Variable	No

5.4.2.4.1. Discriminator

A 12-bit value matching the field of the same name in the Setup Code.

5.4.2.4.2. Vendor ID

A 16-bit value identifying the device manufacturer (see Section 2.5.2, ÒVendor Identifier (Vendor ID, VID)Ó).

5.4.2.4.3. Product ID

A 16-bit value identifying the product (see Product ID).

5.4.2.4.4. Extended Data

Extended Data MAY be made available by commissionable Nodes. This data SHALL be encoded using a standard TLV encoding defined in this section. The location of this data varies based on the NodeÕs commissioning networking technology.

This extended data SHALL be encoded as a TLV structure tagged with an anonymous tag.

The members of this structure SHALL use context-specific tags with the values and meanings shown in the table below.

Tag	Value	Member type	Member Description
RotatingIdTag	0x00	octet string	Rotating Device Identifier

5.4.2.4.5. Rotating Device Identifier

Some device makers need a way to uniquely identify a device before it has been commissioned for vendor-specific customer support purposes. For example, the device maker may need this to iden! tify factory software version and related features, manufacturing date, or to assist in recovery when a setup code has been lost or damaged. In order to avoid privacy issues associated with a fixed unique identifier, devices MAY utilize a Rotating Device Identifier for identification purposes. A Rotating Device Identifier is similar to a serial number but rotates at pre-defined moments.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

The Rotating Device Identifier provides a non-trackable identifier which is unique per-device and that can be used in one or more of the following ways:

- ¥ Provided to the vendorÕs customer support for help in pairing or establishing Node provenance;
- ¥ Used programmatically to obtain a NodeÕs Passcode or other information in order to provide a simplified setup flow. Note that the mechanism(s) by which the Passcode may be obtained is outside of this specification. If the Rotating Device Identifier is to be used for this purpose, the system implementing this feature SHALL require proof of possession by the user at least once before providing the Passcode. The mechanism for this proof of possession, and validation of it, is outside of this specification.

The Rotating Device Identifier is an optional feature for a Node to implement and an optional fea! ture for a Commissioner to utilize. The algorithm used for generating a Rotating Device Identifier SHALL meet the following security and privacy requirements:

1. It SHALL be irreversible in such a way that:
   a. It SHALL prevent recovery of a unique identifier for the device by entities that do not already have access to the set of possible unique identifiers.
   b. Leaking of a common key or equivalent could not be used to recover a unique identifier for all devices sharing the common key.
2. It SHALL protect against long-term tracking by rotating upon each commencement of advertis! ing.
3. It SHALL have a total of at least 64 bits of entropy and SHOULD preferably have more, up to 256 bits.
4. It SHALL NOT contain a fixed identifier such as a serial number.

The Rotating Device Identifier Algorithm below meets these requirements. A Node that implements the Rotating Device Identifier SHALL use either the Rotating Device Identifier Algorithm or a differ! ent algorithm which has been approved and verified by the Connectivity Standards Alliance for this purpose and which meets the same set of security and privacy requirements listed above.

The Rotating Device Identifier Algorithm employs a key derivation algorithm that combines a monotonically increasing lifetime counter with a unique per-device identifier.

The unique identifier SHALL consist of a randomly-generated 128-bit or longer octet string which SHALL be programmed during factory provisioning or delivered to the device by the vendor using secure means after a software update.

The unique identifier SHALL be protected against reading or writing over the air after initial intro! duction into the device, and stay fixed during the lifetime of the device.

The lifetime counter SHALL be an integer at least 16 bits in size, incremented upon each com! mencement of advertising, and wrapping when the maximum value is reached.

The Rotating Device Identifier Algorithm is defined as follows:

```
Rotating Device ID = Rotation Counter ||  Crypto_KDF(
```


Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

```
Ê                    inputKey := Unique ID,
Ê                    salt:= Rotation Counter,
Ê                    info := "RotatingDeviceID",
Ê                    len := 128)
```

(where || is the concatenation operation)

The rotation counter is encoded as 2 bytes using little-endian encoding in the above algorithm, everywhere it appears.

The Rotating Device ID is the concatenation of the current rotation counter and the 16 bytes of the Crypto_KDF result.

5.4.2.4.6. TLV Example

Extended data containing just a Rotating Device Identifier would be encoded as the following bytes:

Offset	Data	Comments
0x00	0x15	Control byte for structure with anonymous tag
0x01	0x30	Control byte for octet string with 1-byte length and a context-specific tag
0x02	0x00	Context-specific tag for Rotating Device Identifier
0x03	0x12	Length of Rotating Device Identifier (e.g. 18 bytes)
0x04	0xXX..0xXX	Rotating Device Identifier
0x16	0x18	End of container

5.4.2.5. Using BLE

This section provides details of how a device announces its commissionable status using BLE tech! nology. Nodes currently commissioned into one or more fabrics SHALL NOT employ this method.

NOTE	Need to add link(s) to BLE specification.

5.4.2.5.1. Device Role

Commissionable devices SHALL implement the role of a Generic Access Profile (GAP) Peripheral.

5.4.2.5.2. Channels

There are three advertising channels used by BLE. All three channels SHOULD be used by commis! sionable devices for BLE advertising.

5.4.2.5.3. Interval

Commissionable devices SHOULD use an Advertising Interval between 20 ms and 60 ms for the first 30 seconds and a value between 150 ms to 1200 ms for the rest of the Announcement duration. Shorter intervals typically result in shorter discovery times.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

5.4.2.5.4. Advertising Mode

Commissionable devices SHALL use the GAP General Discoverable mode, sending connectable undirected advertising events.

5.4.2.5.5. Advertising Address

To ensure privacy, commissionable devices SHALL use LE Random Device Address (see Bluetooth¨ Core Specification 4.2 Vol 6, Part B, Section 1.3.2.1 "Static device address") for BLE Advertising and SHALL change it at least on every boot.

5.4.2.5.6. Advertising Data

In order to reduce 2.4 GHz spectrum congestion due to active BLE scanning, and to extend battery life in battery-powered devices, all critical data used for device discovery is contained in the Adver! tising Data rather than the Scan Response Data. This allows a BLE Commissioner to passively scan (i.e., not issue Scan Requests upon receiving scannable advertisements) and still be able to receive all information needed to commission a device.

Note that if additional vendor-specific information is to be conveyed and does not fit within the Advertising Data, it may be included in the Scan Response Data. See Section 5.4.2.8, ÒManufacturer-specific dataÓ for details on including vendor-specific information.

The following table details the contents of the Advertising PDU payload:

Byte	Value	Description
0	0x02	AD[0] Length == 2 bytes
1	0x01	AD[0] Type == 1 (Flags)
2	0x06	Bit 0 (LE Limited Discoverable Mode) SHOULD be set to 0 Bit 1 (LE General Discoverable Mode) SHOULD be set to 1 If only BLE is supported, this value SHOULD be set to 0x06. If BR/EDR functionality is supported by a commissionable device, this value SHOULD be set accordingly.
3	0x0B	AD[1] Length == 11 bytes
4	0x16	AD[1] Type == 0x16 (Service Data - 16-bit UUID)
5-6	0xFFF6	16-bit Matter UUID assigned by Bluetooth SIG
7	0x00	Matter BLE OpCode == 0x00 (Commissionable) Values 0x01 - 0xFF are reserved
8-9	Variable	Bits[15:12] == 0x0 (Advertisement version) Bits[11:0] == 12-bit Discriminator (see Section 5.4.2.4.1, ÒDiscriminatorÓ)
10-11	Variable	16-bit Vendor ID (see Section 5.4.2.4.2, ÒVendor IDÓ) Set to 0, if elided
12-13	Variable	16-bit Product ID (see Section 5.4.2.4.3, ÒProduct IDÓ) Set to 0, if elided

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Byte	Value	Description
14	Fixed	Bit[0] == Additional Data Flag (see Section 5.4.2.5.7, ÒGATT-based Additional DataÓ) Bits[7:1] are reserved for future use and SHALL be clear (set to 0)

Devices MAY choose not to advertise either the VID and PID, or only the PID due to privacy or other considerations. When choosing not to advertise both VID and PID, the device SHALL set both VID and PID fields to 0. When choosing not to advertise only the PID, the device SHALL set the PID field to 0. A device SHALL NOT set the VID to 0 when providing a non-zero PID.

5.4.2.5.7. GATT-based Additional Data

When the Additional Data Flag is set in the Matter Service Data in the BLE Advertisement, the com! missioner MAY access additional commissioning-related data via an unencrypted read-only GATT characteristic C3 (see Table 31, ÒBTP GATT serviceÓ).

The value of the C3 characteristic SHALL be set to the Extended Data payload of the Discovery Information (see Section 5.4.2.4.4, ÒExtended DataÓ).

5.4.2.6. Using Wi-Fi Temporary Access Points (Soft-AP)

This section details how a device advertises its commissionable state using Wi-Fi Soft-AP functional! ity, wherein the device acts as a Wi-Fi Access Point (AP) that doesnÕt provide Internet access and a Commissioner acts as a Wi-Fi station client and associates with the deviceÕs AP in order to commis! sion it over IPv6. Nodes currently commissioned into one or more fabrics SHALL NOT employ this method.

5.4.2.6.1. Device Role

The device operates as an Access Point, transmitting Beacons and responding to Probe Requests by sending Probe Responses per the rules specified in IEEEÊ802.11-2020.

The Commissioner associates with the DeviceÕs temporary Wi'Fi access point. Once Commissioner and Device have established link-layer connectivity at the Wi'Fi layer, both Commissioner and Device configure themselves unique IPv6 link-local addresses, and then Device Discovery proceeds as for the cases using existing IP-bearing network .

5.4.2.6.2. AP Operating Parameters

The following table specifies the AP operational parameters:

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Parameter	Value
SSID	MATTER-ddd-vvvv-pppp ¥ ddd is the 12-bit Discriminator in hex digits ¥ vvvv is the 16-bit Vendor ID (VID) in hex digits ¥ pppp is the 16-bit Product ID (PID) in hex digits Note that all above elements are present in the QR code for commissioners that require an exact SSID for scanning/connection. NOTE Some devices may choose not to advertise the VID and/or PID due to privacy or other considerations. These devices SHOULD advertise the value 0 instead of the VID/PID.
Hidden SSID	SSID SHALL NOT be hidden as the device may need to be chosen using a mobile OS "network picker" on older mobile OS versions.
BSSID	SHALL be randomly generated on each boot for privacy/tracking reasons. Broadcast bit SHALL be clear, Locally-administered bit SHALL BE set.
Channel	SHALL be chosen from any valid 2.4 GHz ISM channel based on regulatory domain at boot. SHOULD choose randomly from 1, 6, or 11. Vendors may need to choose a specific channel for device-specific reasons.
Security	None
Beacon Interval	100 TUs
DTIM Interval	Not specified (Commissioner power management not expected)

5.4.2.6.3. Matter Vendor-specific IE

This section defines the Information Element (IE) and attributes for Matter devices that support Wi-Fi Soft-AP for commissioning. The Matter IE SHALL be carried in the Wi-Fi Soft-AP Beacon and Probe Response frames.

A Vendor Specific IE format as defined in IEEEÊ802.11-2020 SHALL be used to define the Matter IE in this specification. The format for the Matter IE is shown in Table 45, ÒMatter Information Element formatÓ. Little-endian encoding is used for all fields and subfields in the Matter IE format.

Table 45. Matter Information Element format

Field	Size (Octets)	Value (Hex)	Description
Element ID	1	0xDD	IEEEÊ802.11-2020 vendor specific information element
Length	1	Variable	Length of the following fields in the IE in octets. The Length field is variable and set to 4 plus the total length of the Mat! ter Attributes
OUI	3	4A:19:1B	Connectivity Standards Alliance OUI

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Field	Size (Octets)	Value (Hex)	Description
OUI Type	1	0x00	Identifying the type and version of the Matter IE Values 0x01 - 0xFF are reserved
Matter attributes	Variable	Variable	One or more Matter attributes

The Matter attributes are defined to have a common general format consisting of a one octet Matter attribute identifier field, a one octet Length field, and a variable-length attribute-specific informa! tion field, as shown in Table 46, ÒMatter Attribute formatÓ .

Table 46. Matter Attribute format

Field	Size (Octets)	Value (Hex)	Description
Attribute ID	1	Variable	identifies the type of Matter attribute. Values defined in Table 47, ÒMatter Attribute listÓ .
Length	1	Variable	Length of the following fields in the attribute
Attribute Body	Variable	Variable	Matter attribute specific information fields

The Table 47, ÒMatter Attribute listÓ defines the Matter attributes that SHALL be included in the Wi-Fi Soft-AP Matter IE.

Table 47. Matter Attribute list

Attribute ID (Hex)	Description
0x00	Reserved
0x01	Device OpCode
0x02	Device Information
0x03	Rotating Device Id
0x04 - 0xFF	Reserved

5.4.2.6.3.1. Device State Matter attribute

The format of Device OpCode (Operational Code) Matter attribute is shown in Table 48, ÒDevice State Matter Attribute formatÓ .

Table 48. Device State Matter Attribute format

Field	Size (Octets)	Value (Hex)	Description
Attribute ID	1	0x01	Device OpCode attribute
Length	1	0x01	Length of the following fields in the attribute

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Field	Size (Octets)	Value (Hex)	Description
Attribute Body	1	Variable	0x00 : Commissionable Values 0x01 - 0xFF are reserved

5.4.2.6.3.2. Device Information attribute

The format of Device Information Matter attribute is shown in Table 49, ÒDevice Information Matter Attribute formatÓ .

Table 49. Device Information Matter Attribute format

Field	Size (Octets)	Value (Hex)	Description
Attribute ID	1	0x02	Vendor ID attribute
Length	1	0x06	Length of the following fields in the attribute
Device Discriminator	2	Variable	b0 - b11 : 12-bit discriminator (see Section 5.4.2.4.1, ÒDiscriminatorÓ) b12 - b15 : Reserved, set to zero
VID	2	Variable	16-bit Vendor ID (see Section 5.4.2.4.2, ÒVendor IDÓ) Set to 0, if elided
PID	2	Variable	16-bit Product ID (see Section 5.4.2.4.3, ÒProduct IDÓ) Set to 0, if elided

Devices MAY choose not to advertise either the VID and PID, or only the PID due to privacy or other considerations. When choosing not to advertise both VID and PID, the device SHALL set both VID and PID fields to 0. When choosing not to advertise only the PID, the device SHALL set the PID field to 0. A device SHALL NOT set the VID to 0 when providing a non-zero PID.

5.4.2.6.3.3. Rotating Device Id attribute

The format of Rotating Device Id is shown in Table 50, ÒRotating Device Id Attribute formatÓ .

Table 50. Rotating Device Id Attribute format

Field	Size (Octets)	Value (Hex)	Description
Attribute ID	1	0x03	Rotating Device Id attribute
Length	1	Variable	Length of the following fields in the attribute
RDI	Variable	Variable	Rotating Device Identifier , encoded as a variable length upper-case hexadecimal string, including any leading zeroes.

5.4.2.6.4. Additional Data

Additional data, using the encoding defined above (see Section 5.4.2.4, ÒDiscovery InformationÓ),

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

MAY be included in the Matter IE as an additional attribute, for more information about IE attribute (see Matter Information Element)

5.4.2.6.5. DHCP

A DHCP Server SHALL be implemented on the device if Soft-AP commissioning is used. Though Soft-AP commissioning relies on link-local IPv6 communication, some mobile OSes generate lack-of-con! nectivity warnings to the user if an IPv4 address is not obtained via a DHCP server. The following table specifies the DHCP server operational parameters:

Parameter	Value
Prefix	192.168.226/24 (avoid 192.168.1/24, 192.168.0/24, etc.)
Server IPv4 Address	192.168.226.1
Range	192.168.226.2 to 192.168.226.254.
Lease time	15 minutes (same as discovery timeout)

5.4.2.7. Using Existing IP-bearing Network

This section details how a device that is already connected to an IP-bearing network advertises its commissionable state. The discovery protocols leverage IETF Standard DNS-based Service Discov! ery [RFCÊ6763]. A device SHALL use multicast DNS [RFCÊ6762] on Wi-Fi and Ethernet networks to make itself discoverable. On Thread networks, a device SHALL use the Service Registration Protocol [SRP] and an Advertising Proxy [AdProx] running on a Thread Border Router to make itself discov! erable. Additional details on application of the above protocols in Matter is found in Section 4.3, ÒDiscoveryÓ. The encoding of the information required for discovery during the commissioning process is covered in Section 4.3.1, ÒCommissionable Node DiscoveryÓ .

5.4.2.8. Manufacturer-specific data

If needed, manufacturer-specific data MAY be advertised by a commissionable device using one of the following mechanisms, based on the supported commissioning technology. Commissioners receiving this data SHOULD treat it as opaque unless they have the need to and possess the ability to correctly interpret the information conveyed.

5.4.2.8.1. Using BLE

Any manufacturer-specific data may be included as a Manufacturer Specific Data AD type in the Advertising Data or in the Scan Response data.

Note that to receive Scan Response data information the Commissioner has to perform BLE active scanning that, in addition to creating additional traffic in the shared 2.4 GHz unlicensed band, can delay device discovery and connection, increasing the overall time required to commission a device.

5.4.2.8.2. Using Wi-Fi

Any manufacturer-specific data SHOULD be conveyed using the Vendor-specific Information Ele!

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

ment (IE) mechanism per IEEEÊ802.11-2020 Non-Matter-specific information SHALL NOT be included in the Matter-specified Vendor-specific IE (see Section 5.4.2.6.3, ÒMatter Vendor-specific IEÓ).

5.4.3. Discovery by Commissioner

How a Commissioner discovers a commissionable device depends on the networking technologies that device and the Commissioner supports (see Section 5.4.2.1, ÒTechnology PriorityÓ). Though not all networking technologies must be supported by every device (see Table 36, ÒDiscovery Capabilities BitmaskÓ), a Commissioner SHALL support Commissioning (see Section 5.5, ÒCommissioning FlowsÓ) using existing IP network and over BLE (if having such interface) and SHOULD support commissioning over Wi-Fi Soft-AP.

The following sections detail Commissioner behavior for each of these networking technologies. Though a QR or Manual Pairing code may be scanned or entered prior to discovery, it is not required to do so. However, after scan/entry of the code, the Discriminator, VID and PID elements are available to ensure that the intended device is discovered before proceeding to the connection phase of commissioning.

5.4.3.1. Using BLE

Commissioners SHALL implement the role of a GAP Central. To discover a commissionable device advertising over BLE, a Commissioner SHALL perform a BLE scan across all three advertising channels with a sufficient dwell time, interval, and overall duration of scan. In order to promote quick discovery it is recommended that a Commissioner scan as aggressively as possible within the Commissioner device functionality/UX constraints. In addition, if manufacturer-specific data is not needed, a passive scan (i.e., one that only listens for Advertisement PDUs and does not issue Scan Request PDUs).

If discovery procedure is user initiated the scan interval SHOULD be set between 30 ms and 60 ms, and the scan window SHOULD be set to 30 ms. If discovery procedure is not user initiated (i.e., the Commissioner is scanning in the background), the device may use more relaxed scan, for example, the scan interval set to 1.28 seconds and scan window set to 11.25 ms.

Note: Recommended values are defined in Appendix A: Timers and Constants of Bluetooth¨ Core Specification 4.2 , Vol 3, Part C.

5.4.3.2. Using Wi-Fi

To discover a commissionable device acting as a Soft-AP and advertising its commissionable status, a Commissioner SHALL perform a scan of all 2.4 GHz Wi-Fi channels allowed per its operational regulatory domain. Given that channels 1, 6, and 11 are preferred (see Section 5.4.2.6.2, ÒAP Operating ParametersÓ), scanning of those channels SHOULD be prioritized to minimize discovery time. Where practical and allowed by regulations, active scanning using Probe Requests SHOULD be also be used to help minimize discovery time. However, Commissioners that are always scanning as a background activity SHOULD do so passively (i.e., SHOULD NOT send Probe Requests) in order to reduce unnecessary transmissions in the shared 2.4 GHz spectrum.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

5.4.3.3. Using Existing IP-bearing Network

To discover a commissionable device over an existing IP-bearing network connection, the Commissioner shall perform service discovery using DNS-SD as detailed in Section 4.3, ÒDiscoveryÓ, and more specifically in Section 4.3.1, ÒCommissionable Node DiscoveryÓ.

5.5. Commissioning Flows

There are two commissioning flows depending upon the networking capability of the Commissioner and Commissionee, namely concurrent connection commissioning flow and non-concurrent connection commissioning flow.

A Commissioner and Commissionee with concurrent connection have the ability to maintain two network connections simultaneously. One connection is between the Commissioner (or Commissionee) and the operational network (e.g., home Wi-Fi network or Thread network) that the Commissionee is being programmed to join. The second connection is between the Commissioner and Commissionee for commissioning as is referred to as commissioning channel. A Commissioner and Commissionee with non-concurrent connection capability cannot be simultaneously connected to both the operational network that the Commissionee is being configured to join, and the commissioning channel.

The two connections MAY either be on the same or on different networking interfaces. For example, a Commissioner uses its Wi-Fi interface to connect to the operational network, but use its Bluetooth Low Energy interface for commissioning.

To determine whether a Commissionee has concurrent or non-concurrent connection capability, the Commissioner can use the SupportsConcurrentConnection attribute of the General Commissioning Cluster.

Commissioning SHALL be a time-bound process that completes before expiration of a fail-safe timer. The fail-safe timer SHALL be set at the beginning of commissioning. If the fail-safe timer expires prior to commissioning completion, the Commissioner and Commissionee SHALL terminate commissioning. Successful completion of commissioning SHALL disarm the fail-safe timer.

A Commissionee that is ready to be commissioned SHALL accept the request to establish a PASE session with the first Commissioner that initiates the request. When a Commissioner is either in the process of establishing a PASE session with the Commissionee or has successfully established a session, the Commissionee SHALL NOT accept any more requests for new PASE sessions until session establishment fails or the successfully established PASE session is terminated on the commissioning channel (see CloseSession in Secure Channel Status Report Messages). In the event a CloseSession status message is sent or received:

1. If the fail-safe timer is armed, the fail-safe timer SHALL be considered expired and the cleanup steps detailed in Section 11.9.7.2, ÒArmFailSafe CommandÓ SHALL be executed.
2. If the commissioning window is still open, the Commissionee SHALL continue listening for commissioning requests.

In order to avoid locking out the Commissionee from accepting new PASE session requests indefinitely, a Commissionee SHALL expect a PASE session to be established within 60 seconds of receiv!

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

ing the initial request. This means the Commissionee SHALL expect to receive the PAKE3 message within 60 seconds after sending a PBKDFParamResponse in response to a PBKDFParamRequest message from the Commissioner to establish a PASE session. If the PASE session is not established within the expected time window the Commissionee SHALL terminate the current session estab! lishment using the INVALID_PARAMETER status code as described in Section 4.10.1.3, ÒSecure Channel Status Report MessagesÓ.

The commissioning commands and attributes are defined in Clusters (see Section 11.8, ÒNetwork Commissioning ClusterÓ , Section 11.9, ÒGeneral Commissioning ClusterÓ , Section 11.13, ÒThread Net! work Diagnostics ClusterÓ , and Section 11.14, ÒWi-Fi Network Diagnostics ClusterÓ) and are sent, written, or read using the Interaction Model (see Interaction Model).

Figure 30, ÒConcurrent connection commissioning flowÓ and Figure 31, ÒNon-concurrent connec! tion commissioning flowÓ depict the commissioning flow between the Commissioner and Commis! sionee with concurrent connection ability and non-concurrent connection ability, respectively. The specific steps are described below. Unless indicated otherwise, a commissioner SHALL complete a step, including waiting for any responses to commands it sends in that step, before moving on to the next step.

1. The Commissioner initiating the commissioning SHALL have regulatory and fabric information available, and SHOULD have accurate date, time and timezone.
2. Commissioner and Commissionee SHALL find each other over networking interfaces such as Bluetooth, Wi-Fi, or Ethernet using the process of discovery and establish a commissioning channel between each other (see Section 5.4, ÒDevice DiscoveryÓ).
3. Commissioner and Commissionee SHALL establish encryption keys with PASE (see Section 4.13.1, ÒPasscode-Authenticated Session Establishment (PASE)Ó) on the commissioning channel. All subsequent messages on the commissioning channel are encrypted using PASE-derived encryption keys. Upon completion of PASE session establishment, the Commissionee SHALL autonomously arm the Fail-safe timer for a timeout of 60 seconds. This is to guard against the Commissioner aborting the Commissioning process without arming the fail-safe, which may leave the device unable to accept additional connections.
4. Commissioner SHALL re-arm the Fail-safe timer on the Commissionee to the desired commis! sioning timeout within 60 seconds of the completion of PASE session establishment, using the ArmFailSafe command (see Section 11.9.7.2, ÒArmFailSafe CommandÓ). A Commissioner MAY obtain device information including guidance on the fail-safe value from the Commissionee by reading BasicCommissioningInfo attribute (see Section 11.9.6.2, ÒBasicCommissioningInfo AttributeÓ) prior to invoking the ArmFailSafe command.
5. Commissioner SHALL configure regulatory information in the Commissionee if it has at least one instance of Network Commissioning cluster on any endpoint with either the WI (i.e. Wi-Fi) or TH (i.e. Thread) feature flags set in its FeatureMap. Commissioner SHOULD configure UTC time, timezone, and DST offset, if the Commissionee supports the time cluster. The order of con! figuration of this information is not critical. The UTC time is configured using SetUtcTime com! mand (see Section 11.16.7.1, ÒSetUtcTime CommandÓ) while timezone and DST offset are set through TimeZone (see Section 11.16.6.6, ÒTimeZone AttributeÓ) and DstOffset attribute (see Sec! tion 11.16.6.7, ÒDstOffset AttributeÓ), respectively. The regulatory information is configured using SetRegulatoryConfig (see Section 11.9.7.4, ÒSetRegulatoryConfig CommandÓ).

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

6. Commissioner SHALL establish the authenticity of the Commissionee as a certified Matter device (see Section 6.2.3, ÒDevice Attestation ProcedureÓ).

   %If the Commissionee fails the Device Attestation Procedure, for any reason, the Commissioner MAY choose to either continue to the Commissioning, or terminate it, depending on implementation-dependent policies.

   %Upon failure of the procedure, the Commissioner SHOULD warn the user that the Commissionee is not a fully trusted device, and MAY give the user the choice to authorize or deny the commissioning. Such a warning enables user choice in Commissionee trust on their Fabric, for development workflows, as well as homebrew device development. Such a warning SHOULD contain as much information as the commissioner can provide about the Commissionee, and SHOULD be adapted to the reason of the failure, for example by being different between the case of an expired certificate and the case of a failed signature verification.

   %Reasons for failing the Device Attestation procedure MAY include, but are not limited to, the following:

   & The Commissionee being of a device type currently in development or not yet certified (see certification_type in the Certification Declaration).

   & The CommissioneeÕs PAA not being in the CommissionerÕs trusted set.

   & The Commissioner having obtained knowledge that a PAA or PAI certificate presented has been revoked, or that the particular Device Attestation Certificate has been revoked.

   & The Commissionee failing to prove possession of the Device Attestation private key, either by programming error, malicious intent or other reasons.

   & One of the elements of the CommissioneeÕs Device Attestation Certificate chain not meeting the policy validation steps of the Device Attestation Procedure, including errors on validity period.

   %If a Commissioner denies commissioning for any reason, it SHOULD notify the user of the reason with sufficient details for the user to understand the reason, so that they could determine if it would be possible to commission the device using a different Commissioner.

7. Following the Device Attestation Procedure yielding a decision to proceed with commissioning, the Commissioner SHALL request operational CSR from Commissionee using the CSRRequest command (see Section 11.17.7.5, ÒCSRRequest CommandÓ). The CSRRequest command will cause the generation of a new operational key pair at the Commissionee.

8. Commissioner SHALL generate or otherwise obtain an Operational Certificate containing Operational ID after receiving the CSRResponse command from the Commissionee (see Section 11.17.7.5, ÒCSRRequest CommandÓ), using implementation-specific means.

9. Commissioner SHALL install operational credentials (see Figure 38, ÒNode Operational Credentials flowÓ) on the Commissionee using the AddTrustedRootCertificate and AddNOC commands.

10. Commissioner MAY configure the Access Control List (see Access Control Cluster) on the Commissionee in any way it sees fit, if the singular entry added by the AddNOC command in the previous step granting Administer privilege over CASE authentication type for the Node ID provided with the command is not sufficient to express its desired access control policies.

11. If the Commissionee both supports it and requires it, the Commissioner SHALL configure the operational network at the Commissionee using commands such as AddOrUpdateWiFiNetwork

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

(see Section 11.8.8.4, ÒAddOrUpdateWiFiNetwork CommandÓ) and AddOrUpdateThreadNetwork (see Section 11.8.8.5, ÒAddOrUpdateThreadNetwork CommandÓ). A Commissionee requires network commissioning if it is not already on the desired operational network. A Commissionee supports network commissioning if it has any NetworkCommissioning cluster instances. A Commissioner MAY learn about the networks visible to the Commissionee using ScanNetworks command (see Section 11.8.8.2, ÒScanNetworks CommandÓ).

12. The Commissioner SHALL trigger the Commissionee to connect to the operational network using ConnectNetwork command (see Section 11.8.8.10, ÒConnectNetwork CommandÓ) unless the Commissionee is already on the desired operational network.
13. Finalization of the Commissioning process begins. An Administrator configured in the ACL of the Commissionee by the Commissioner SHALL use Operational Discovery to discover the Commissionee. This Administrator MAY be the Commissioner itself, or another Node to which the Commissioner has delegated the task.
14. The Administrator SHALL open a CASE (see Section 4.13.2, ÒCertificate Authenticated Session Establishment (CASE)Ó) session with the Commissionee over the operational network.
15. The Administrator having established a CASE session with the Commissionee over the operational network in the previous steps SHALL invoke the CommissioningComplete command (see Section 11.9.7.6, ÒCommissioningComplete CommandÓ). A success response after invocation of the CommissioningComplete command ends the commissioning process.

While the Administrator of steps 13-15 will, in many situations, be the Commissioner Node itself, it MAY be a different Node that was configured by the Commissioner to have Administer privilege against the CommissioneeÕs General Commissioning Cluster . This is to support flexibility in finalizing the Commissioning. From a CommissioneeÕs perspective, all Nodes with Administer privilege in the CommissioneeÕs ACL are equivalent once the Node has a Node Operational Certificate and associated Node Operational Identifier on the Fabric into which it was just commissioned.

A Commissioner MAY configure UTC time, Operational ID, and Operational certificates, etc., information over an arbitrary number of interactions at the Commissionee, over the operational network after the commissioning is complete, or over the commissioning channel after PASE-derived encryption keys are established during commissioning.

In concurrent connection commissioning flow the commissioning channel SHALL terminate after successful step 15 (CommissioningComplete command invocation). In non-concurrent connection commissioning flow the commissioning channel SHALL terminate after successful step 12 (trigger joining of operational network at Commissionee). The PASE-derived encryption keys SHALL be deleted when commissioning channel terminates. The PASE session SHALL be terminated by both Commissioner and Commissionee once the CommissioningComplete command is received by the Commissionee.

In both concurrent connection commissioning flow and non-concurrent connection commissioning flow, the Commissioner MAY choose to continue commissioning and override the failure in step 6 (Commissionee attestation).

5.5.1. Commissioning Flows Error Handling

Overall, all Commissioning operations employ actions using cluster attributes and commands that

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

are also, in certain cases, available during normal steady-state operation once commissioned.

Whenever the Fail-Safe timer is armed, Commissioners and Administrators SHALL NOT consider any cluster operation to have timed-out before waiting at least 30 seconds for a valid response from the cluster server. Some commands and attributes with complex side-effects MAY require longer and have specific timing requirements stated in their respective cluster specification.

Some request commands used for Commissioning and administration have a 'Breadcrumb' argu! ment. When set, this argument SHALL be used to update the value of the Breadcrumb Attribute as a side-effect of successful execution of those commands. On command failures, the Breadcrumb Attribute SHALL remain unchanged.

In concurrent connection commissioning flow, the failure of any of the steps 2 through 10 SHALL result in the Commissioner and Commissionee returning to step 2 (device discovery and commis! sioning channel establishment) and repeating each step. The failure of any of the steps 11 through 15 in concurrent connection commissioning flow SHALL result in the Commissioner and Commis! sionee returning to step 11 (configuration of operational network information). In the case of fail! ure of any of the steps 11 through 15 in concurrent connection commissioning flow, the Commis! sioner and Commissionee SHALL reuse the existing PASE-derived encryption keys over the commis! sioning channel and all steps up to and including step 10 are considered to have been successfully completed.

In non-concurrent connection commissioning flow, the failure of any of the steps 2 through 15 SHALL result in the Commissioner and Commissionee returning to step 2 (device discovery and commissioning channel establishment) and repeating each step.

Commissioners that need to restart from step 2 MAY immediately expire the fail-safe by invoking the ArmFailSafe command with an ExpiryLengthSeconds field set to 0. Otherwise, Commissioners will need to wait until the current fail-safe timer has expired for the Commissionee to begin accept! ing PASE again.

In both concurrent connection commissioning flow and non-concurrent connection commissioning flow, the Commissionee SHALL exit Commissioning Mode after 20 failed attempts.

Once a Commissionee has been successfully commissioned by a Commissioner into its fabric, the commissioned Node SHALL NOT accept any more PASE requests until any one of the following con! ditions is met:

¥ Device is factory-reset.

¥ Device enters commissioning mode.

Ongoing administration of Nodes by Administrators employs many of the same clusters and con! straints related to Fail-Safe timer and cluster operation time-outs used for initial or subsequent Commissioning into new Fabrics. The respective cluster specifications for the Node Operational Credentials Cluster and the Network Commissioning Cluster reflect the necessary usage of the Arm! FailSafe and CommissioningCompletecommands of the General Commissioning Cluster to achieve consistent state during administrative operations.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

5.5.2. Commissioning Flow Diagrams

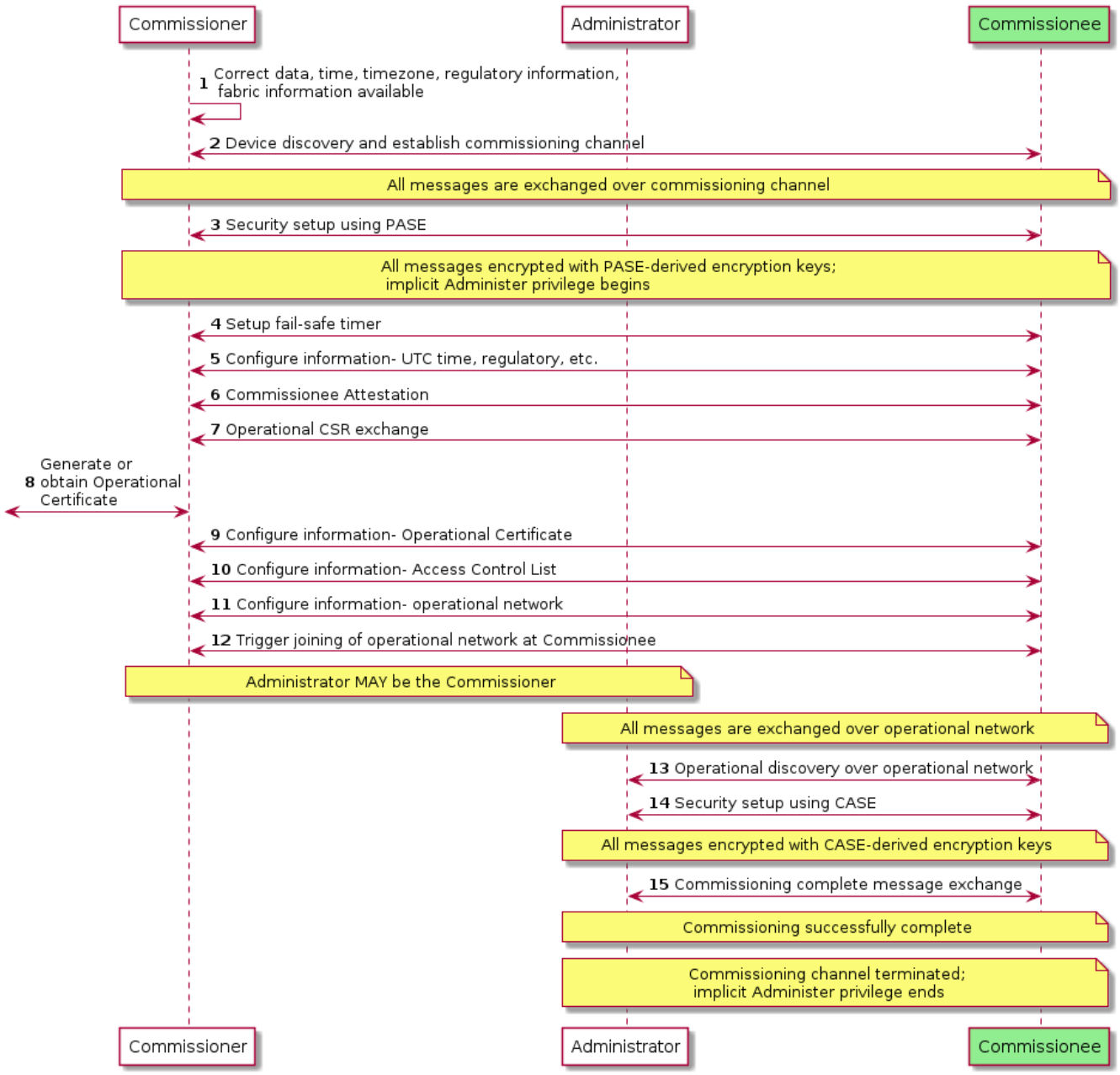


Figure 30. Concurrent connection commissioning flow

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

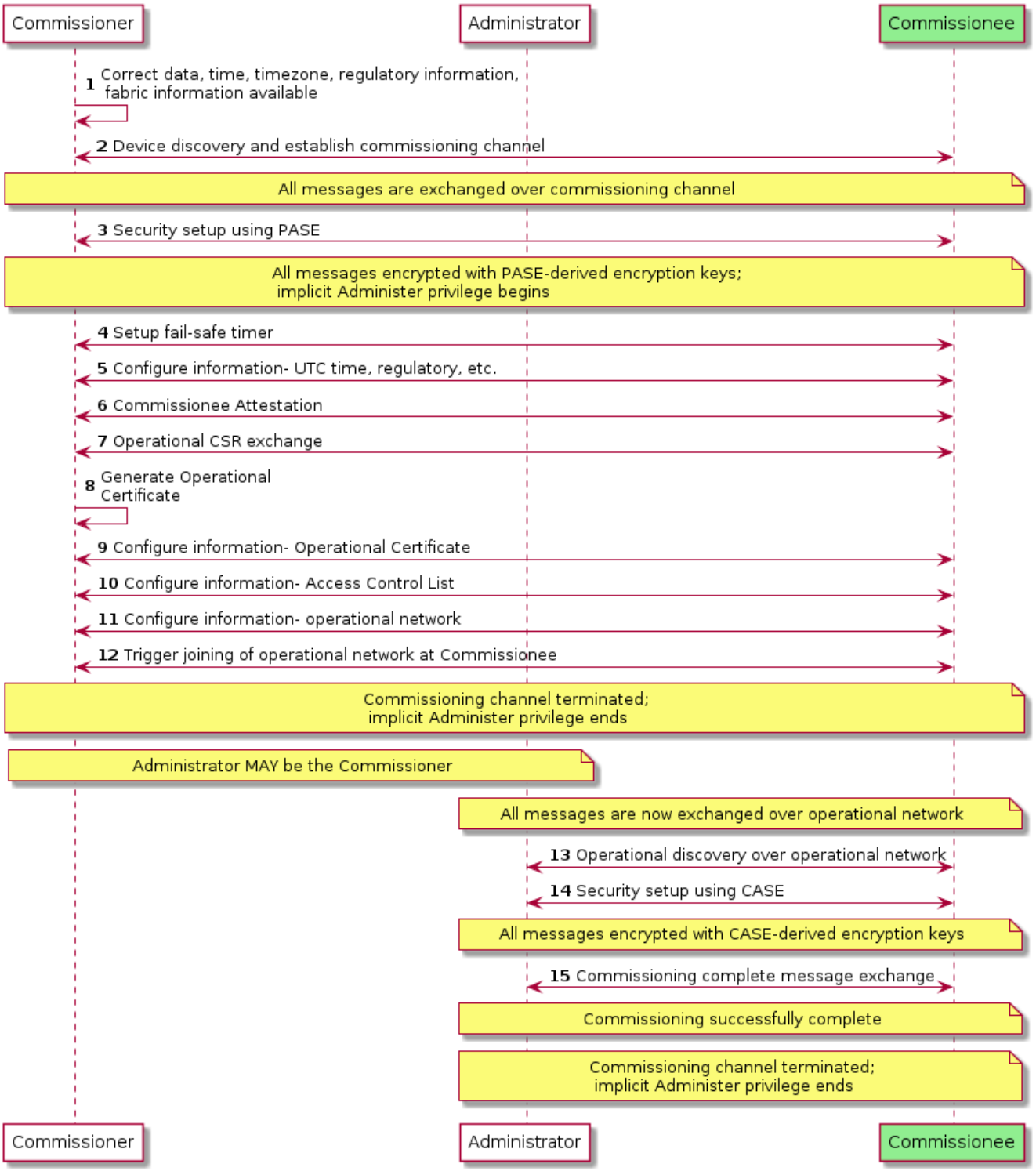


Figure 31. Non-concurrent connection commissioning flow

5.6. Administrator Assisted Commissioning Flows

5.6.1. Introduction

In this method, a current Administrator of a Node first sends the Open Commissioning Window command to the Node over a CASE session. The new Administrator MUST already have network connectivity and complete commissioning based on the two flows described below.

The commands for these flows are defined in Section 11.18, ÒAdministrator Commissioning Clus! terÓ.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

5.6.2. Basic Commissioning Method (BCM)

This method is OPTIONAL for Nodes and Administrators/Commissioners to implement. In this method, the current Administrator MUST send the Open Basic Commissioning Window command to the Node over a CASE session. The Node SHALL advertise its presence over DNS-SD (see Section 5.4.2.7, ÒUsing Existing IP-bearing NetworkÓ and Commissionable Node Discovery) after receiving the Open Basic Commissioning Window command.

The new AdministratorÕs Commissioner then completes commissioning with the Node using similar Commissioning flow as it would do for a factory-new device (although note that IP channel is used for discovery). It can either scan the QR code format or use the Manual Pairing Code format of the Section 5.1, ÒOnboarding PayloadÓ of the Node.

The following steps describe a possible sequence of events for BCM commissioning:

1. Current Administrator puts the Node in Open Basic Commissioning Window for a specified time window, and receives success response from the Node on the Open Basic Commissioning Win! dow command.
   a. When the targeted Node is a SED, the current Administrator can guide the user to perform some action to 'wake' the device from its sleep cycle.
2. New Administrator completes commissioning within the prescribed window using steps out! lined in Figure 30, ÒConcurrent connection commissioning flowÓ .

5.6.3. Enhanced Commissioning Method (ECM)

This method is MANDATORY for Nodes and Commissioners/Administrators to implement. When using ECM, the NodeÕs current Administrator instructs the Node over a CASE session, to go into Open Commissioning Window . It SHALL choose a new RANDOM passcode and SHALL compute and send the corresponding PAKE passcode verifier to the Node. Actual value of the passcode SHALL NOT be sent to the Node. The current Administrator then presents the new passcode and discrimi! nator as described below . The Node SHALL advertise its presence over DNS-SD (see Section 5.4.2.7, ÒUsing Existing IP-bearing NetworkÓ and Commissionable Node Discovery) after receiving the Open Commissioning Window command. Sleepy Nodes SHOULD include the optional SII key in their TXT advertisement.

5.6.3.1. Presentation of Onboarding Payload for ECM

Presentation of the passcode and other relevant information SHALL be done at least with one or more of the methods below, depending on the capabilities of the first Administrator opening the OCW:

1. If a user interface display is supported, the temporary Onboarding Payload SHALL be displayed using a textual representation of the Manual Pairing Code, using the 11-digit variant: it SHALL NOT contain the VENDOR_ID or PRODUCT_ID as the onboarding of the node(s) using the ECM cannot be subject to User-Intent or Custom Flows.
2. If a user interface display is supported, the temporary Onboarding Payload SHOULD also be dis! played using the definitions included in Section 5.1.3, ÒQR CodeÓ subject to the following con! straints:

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

a. If only a single Node is being subjected to the ECM, the Vendor ID and Product ID in the onboarding payload SHALL be the same as those of that Node.

b. If multiple Nodes are being subjected to the ECM using the same onboarding payload , the Vendor ID SHALL be set to 0x0000 (Matter Standard) and the Product ID SHALL be set to 0x0000 (consistent with the value used for not advertising a Product ID in Device Announcement) .

c. The Custom Flow element SHALL be set to 0 to indicate standard flow.

d. The Discovery Capabilities Mask SHALL have ONLY bit 2 set to indicate the Node is only discoverable on the IP network.

e. The Passcode element SHALL be set by the existing Administrator to the same value as the passcode chosen for this ECM operation.

f. The Discriminator element SHALL be set by the existing Administrator to the same value as the Discriminator parameter in Section 11.18.8.1, ÒOpenCommissioningWindow (OCW) CommandÓ.

g. If multiple Nodes are subjected to ECM, the Section 5.1.5, ÒTLV ContentÓ SHALL contain an entry with kTag_NumberofDevices containing the number of devices that are expected to participate in the onboarding with this ECM operation.

h. When the Commissioning Timeout parameter of the OCW command is set to less than the allowed maximum (15 minutes), the Section 5.1.5, ÒTLV ContentÓ SHALL contain an entry with kTag_CommissioningTimeout containing the value of the Commissioning Timeout parameter used for this ECM operation.

3. If only audio output is supported, the temporary Onboarding Payload SHALL be delivered using a voice prompt of the Manual Pairing Code format. A method SHOULD be available for the user to have the pairing code repeated.

Remote UIs, both visual and audio$Ñ$such as a manufacturer-specific mobile app or a web UI$Ñ$are expressly permitted in the set of acceptable mechanisms for conveyance of the onboarding information.

This method allows a current Administrator to set multiple Nodes for commissioning with a new administrator with an appropriate Commissioning Window , by turning on Open Commissioning Window and sending the PAKE passcode verifier to a series of Nodes. The new Administrator uses the information in Manual Pairing Code to discover the Nodes that are in Commissioning mode and commission them using the new passcode.

The following steps describe a possible sequence of events for ECM commissioning:

1. Current Administrator puts the Node(s) in commissioning mode for a specified time window with a new setup passcode, and receives success responses from the involved Node(s) on the Open Commissioning Window command.

   a. When one or more SED are among the targeted Nodes, the current Administrator can guide the user to perform some action to 'wake' these devices from their sleep cycle.

2. Current Administrator presents Onboarding Payload as described above .

3. New Administrator completes commissioning within the prescribed window using steps out!

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

lined in Figure 30, ÒConcurrent connection commissioning flowÓ .

5.6.4. Open Commissioning Window

The following sequence diagram shows steps current Administrator takes to enable Open Commissioning Window .

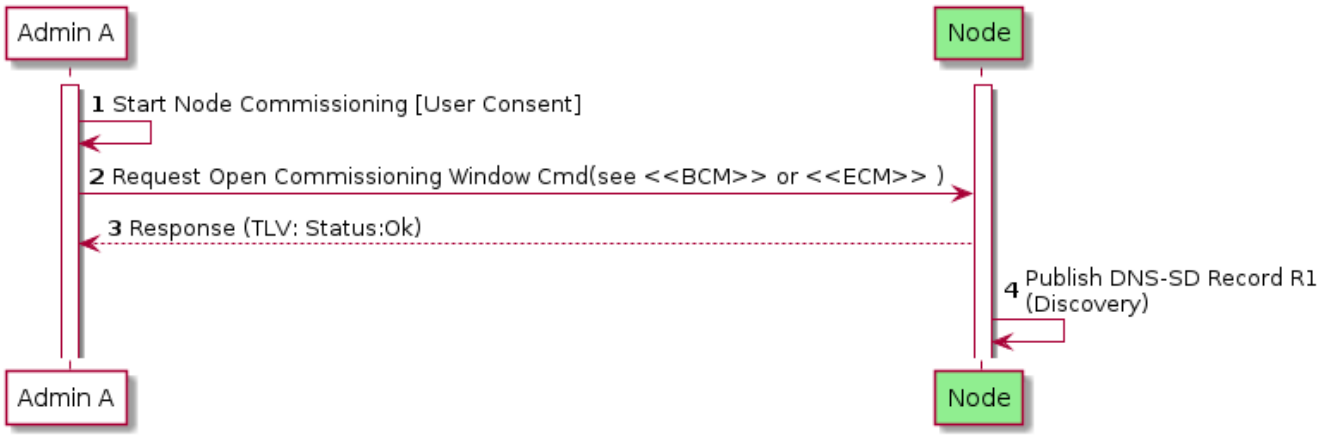


Figure 32. Open Commissioning Window (Administrator A)

5.7. Device Commissioning Flows

This section describes the three different flows for out-of-box commissioning that a Matter device manufacturer may select for a given product. For each flow, a description is provided which includes actions required to place the device into commissioning mode, fields in the Onboarding Payload which identify the flow selected by the device manufacturer, fields in the Distributed Compliance Ledger that provide information used for commissioning and help the Commissioner provide the user with appropriate instructions, and requirements for device packaging relating to the Onboarding Payload. The three flows are the following:

- ¥ Standard Commissioning Flow
- ¥ User-Intent Commissioning Flow
- ¥ Custom Commissioning Flow

Matter device manufacturers SHALL use the Distributed Compliance Ledger to provide commissioners with information and instructions for both initial and secondary commissioning, and SHOULD use this Ledger to provide links to the user guide, a link to a manufacturer app, and other pre-setup information, to enable an optimal commissioning flow without requiring bilateral arrangements between each commissioner manufacturer and each device manufacturer.
Some fields in the Ledger SHALL or SHOULD be populated, depending on the type of commissioning flow, as detailed in the text below and in the Distributed Compliance Ledger section.

5.7.1. Standard Commissioning Flow

- ¥ A Standard Commissioning Flow device SHALL be available for initial commissioning by any Matter commissioner.
- ¥ A Standard Commissioning Flow device, when in factory-new state, SHALL start advertising

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

automatically upon power on (see Commencement).

- A Standard Commissioning Flow device SHALL set Custom Flow bits in the Onboarding Payload to indicate '0 - Standard Flow'.
- A Standard Commissioning Flow device SHALL follow the rules for Manual Pairing Code and QR Code Inclusion .
- For the case where the device has stopped advertising (e.g. user has powered on the device longer ago than the advertisement period), the manufacturer SHOULD provide guidance about how to bring the device back into advertising mode using the CommissioningModeInitial! StepsHint field from the Distributed Compliance Ledger . Commissioners SHOULD use this infor! mation to guide the user for this case.
- When commissioning fails, the commissioner MAY also reference Distributed Compliance Ledger fields such as UserManualUrl, SupportUrl and ProductUrl to assist the user in further steps to resolve the issue(s).
- The Distributed Compliance Ledger entries for Standard Commissioning Flow devices SHALL include the CommissioningCustomFlow field set to '0 - Standard' and the CommissioningMod! eInitialStepsHint field set to a non-zero integer value, with bit 0 (Power Cycle) being set to 1. The CommissioningModeInitialStepsInstruction field SHALL be set when CommissioningModeIni! tialStepsHint has a Pairing Instruction dependency.

Table 51. Values of Ledger fields to represent Standard Commissioning Flow

Field Name	Value(s)
CommissioningCustomFlow	0 - Standard (Mandatory)
CommissioningModeInitialStepsHint	This field SHALL be set to a non-zero integer value. See Pairing Hint Table for a complete list of pairing instructions. Example value: 33 - The following bits are set: 0 (Power Cycle - Mandatory), 5 (Device Manual - Optional). Bit 1 (Device Manufacturer URL) MAY be set.
CommissioningModeInitialStepsInstruction	The field SHALL be set when Commissioning! ModeInitialStepsHint has a Pairing Instruction dependency. See PI Dependency column of Pair! ing Hint Table to determine which pairing hints have Pairing Instruction dependency and there! fore require this field to be populated.

5.7.2. User-Intent Commissioning Flow

- A User-Intent Commissioning Flow device SHALL be available for initial commissioning by any Matter commissioner.
- A User-Intent Commissioning Flow device, when in factory-new state, SHALL NOT start adver! tising automatically upon application of power (see Commencement).
- To place a User-Intent Commissioning Flow device into advertising mode, some form of user

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

interaction with the device beyond application of power is required (see Pairing Hint Table). If a Device Manufacturer setup artifact is required for this, beyond documentation, then the device is a Custom Commissioning Flow device and not a User-Intent Commissioning Flow device. The documentation MAY be printed or in the form of online documentation (e.g. Section 11.22.5.8, ÒUserManualUrlÓ).

- A User-Intent Commissioning Flow device SHALL follow the rules for Manual Pairing Code and QR Code Inclusion .
- The Distributed Compliance Ledger entries for User-Intent Commissioning Flow devices SHALL include the CommissioningCustomFlow field set to '1 - User Intent' and the CommissioningMod! eInitialStepsHint field set to a non-zero integer value. Bit 0 (Power Cycle) in the Commissioning! ModeInitialStepsHint field SHALL be set to 0. The CommissioningModeInitialStepsInstruction field SHALL be set when CommissioningModeInitialStepsHint has a Pairing Instruction depen! dency.
- A User-Intent Commissioning Flow device SHALL set Custom Flow bits in the Onboarding Pay! load to indicate '1 - User Intent'.
- The commissioner SHOULD reference Distributed Compliance Ledger fields such as Commis! sioningModeInitialStepsHint , CommissioningModeInitialStepsInstruction , UserManualUrl, and SupportUrl to assist the user during commissioning, e.g. to explain how to bring the device into commissioning mode.

Table 52. Values of Ledger fields to represent User-Intent Commissioning Flow

Field Name	Value(s)
CommissioningCustomFlow	1 - User Intent (Mandatory)
CommissioningModeInitialStepsHint	This field SHALL be set to a non-zero integer value. See Pairing Hint Table for a complete list of pairing instructions. Example value: 96 - The following bits are set: 6 (Press Reset Button - Optional), 5 (Device Man! ual - Optional). Bit 1 (Device Manufacturer URL) MAY be set.
CommissioningModeInitialStepsInstruction	The field SHALL be set when Commissioning! ModeInitialStepsHint has a Pairing Instruction dependency. See PI Dependency column of Pair! ing Hint Table to determine which pairing hints have Pairing Instruction dependency and there! fore require this field to be populated.

5.7.3. Custom Commissioning Flow

- A Custom Commissioning Flow device SHALL require interaction with custom steps, guided by a service provided by the manufacturer for initial device setup, before it can be commissioned by any Matter commissioner.
- A Custom Commissioning Flow device MAY include the Onboarding Payload on-device or in

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

packaging. If it is not included on the device or in packaging, then it SHALL be provided to the user through other means provided by the manufacturer.

¥ A Custom Commissioning Flow device SHALL set Custom Flow bits in the Onboarding Payload to indicate '2 - Custom'.

¥ The Distributed Compliance Ledger entries for Custom Commissioning Flow devices SHALL include:

%the CommissioningCustomFlow field set to '2 - Custom'

%the CommissioningModeInitialStepsHint with bit 0 (Power Cycle) set to 0 and bit 1 (Device Manufacturer URL) set to 1

%the CommissioningCustomFlowUrl field populated in order to indicate to commissioners that initial commissioning can only be completed by the user visiting the URL contained therein.
This URL will typically lead to a web page with relevant instructions and/or to a server which (e.g. by looking at the User-Agent) redirects the user to allow viewing, downloading, installing or using a manufacturer-provided means for guiding the user through the process and bring the device into a state that it is available for commissioning by any commissioner.
Since the URL is retrieved from a DCL entry corresponding to a specific VID and PID combi! nation, the device manufacturer MAY choose to use any constructed URL valid in a HTTP GET request (i.e. dedicated for the product the user wants to commission) such as, for exam! ple, https://www.example.com/download-install-app?vid=FFF1&pid=1234 . All HTTP based URLs SHALL use the https scheme.

¥ When a Commissioner encounters a device with Custom Flow field (in Onboarding Payload) or its CommissioningCustomFlow field (in Distributed Compliance Ledger) set to '2 - Custom', it SHOULD use the CommissioningCustomFlowUrl to guide the user on how to proceed, unless it has alternative means to guide the user to successful commissioning.

%If a Commissioner follows or launches the CommissioningCustomFlowUrl after a User request, it SHALL expand it as described in Section 5.7.3.1, ÒCommissioningCustomFlowUrl formatÓ.

¥ A manufacturer contemplating using this flow should realize that

%This flow typically requires internet access to access the URL, so initial commissioning of the device may fail if there is no internet connection at that time/location.

%If the flow requires an app, it needs to be made available for popular platforms amongst the user population; some of their platforms running a commissioner (e.g. a smart speaker not running a popular mobile OS) may thus not be able to be used for the initial commissioning of such devices.

Table 53. Values of Ledger fields to represent Custom Commissioning Flow

Field Name	Value(s)
CommissioningCustomFlow	2 - Custom (Mandatory)
CommissioningCustomFlowUrl	'URL' - Device Manufacturer URL (Mandatory)

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Field Name	Value(s)
CommissioningModeInitialStepsHint	This field SHALL be set to a non-zero integer value with at least bit 1 set (Device Manufacturer URL). See Pairing Hint Table for a complete list of pairing instructions. Example value: 2 - The following bits are set: 1 (Device Manufacturer URL) (Mandatory).
CommissioningModeInitialStepsInstruction	The field SHALL be set when CommissioningModeInitialStepsHint has a Pairing Instruction dependency. See PI Dependency column of Pairing Hint Table to determine which pairing hints have Pairing Instruction dependency and therefore require this field to be populated.

5.7.3.1. CommissioningCustomFlowUrl format

The CommissioningCustomFlowUrl MAY contain a query component (see RFC 3986 section 3.4). If a query is present, it SHALL be composed of one or more key-value pairs:

- The query SHALL use the & delimiter between key/value pairs.
- The key-value pairs shall in the format name=<value> where name is the key name, and <value> is the contents of the value encoded with proper URL-encoded escaping.
- If key MTcu is present, it SHALL have a value of "_" (i.e. MTcu=_). This is the "callback URL (CallbackUrl) placeholder".
- If key MTop is present, it SHALL have a value of "_" (i.e. MTop=_). This is the "onboarding payload placeholder".
- Any key whose name begins with MT not mentioned in the previous bullets SHALL be reserved for future use by this specification. Manufacturers SHALL NOT include query keys starting with MT in either the CommissioningCustomFlowUrl or CallbackUrl unless they are referenced by a version of this specification.

When the CommissioningCustomFlowUrl for a Custom Commissioning Flow device includes the MTop key, the Passcode embedded in any Onboarding Payload placed on-device or in packaging SHALL NOT be one that can be used for secure channel establishment with the device. This requirement is intended to ensure a shared secret used for proof of possession will not be transferred to a server without user consent. A Custom Commissioning Flow device MAY utilize Onboarding Payload fields such as the Serial Number (see kTag_SerialNumber) to pass device identification to the server specified in CommissioningCustomFlowUrl, as these fields by themselves could not be used to gain access to the device on their own like the Passcode could.

When the CommissioningCustomFlowUrl for a Custom Commissioning Flow device includes the MTop key, the Passcode embedded in any Onboarding Payload placed on-device or in packaging MAY be set to 0 in order to provide a hint to the Commissioner that it is not one that can be used for secure channel establishment with the device. This would allow the Commissioner to avoid attempting to commission the device if an advertisement from it is detected.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Any other element in the CommissioningCustomFlowUrl query field not covered by the above rules, as well as the fragment field (if present), SHALL remain as obtained from the Distributed Compliance Ledger 's CommissioningCustomFlowUrl field, including the order of query key/value pairs present.

5.7.3.1.1. Expansion of CommissioningCustomFlowUrl by Commissioner

Once the URL is obtained, it SHALL be expanded to form a final URL (ExpandedCommissioningCustomFlowUrl) by proceeding with the following substitution algorithm on the original CommissioningCustomFlowUrl :

1. If key MTcu is present, compute the CallbackUrl desired (see Section 5.7.3.2, ÒCallbackUrl format for Custom Commissioning Flow responseÓ), and substitute the placeholder value "_" (i.e. in MTcu=_) in the CommissioningCustomFlowUrl with the desired contents, encoded with proper URL-encoded escaping (see RFCÊ3986 section 2).
2. If key MTop is present, substitute the the placeholder value "_" (i.e. in MTop=_) in the CommissioningCustomFlowUrl with either numeric manual code , or QR code body including the MT: prefix and TLV data (if present), encoded with proper URL-encoded escaping (see RFCÊ3986 section 2). Note that both methods SHOULD be supported by the ManufacturerÕs custom flow.

A Commissioner SHALL NOT append the MTop= query key/value pair unless the key/value pair was already present as MTop=_ in the CommissioningCustomFlowUrl previously obtained. This constraint enables the determination of which products make use of the payload in their Custom Commissioning Flow infrastructure by inspection of the Distributed Compliance Ledger records.

The final URL after expansion (ExpandedCommissioningCustomFlowUrl) SHALL be the one to follow per Section 5.7.3, ÒCustom Commissioning FlowÓ , rather than the original value obtained from the Distributed Compliance Ledger.

5.7.3.2. CallbackUrl format for Custom Commissioning Flow response

If a CallbackUrl field (i.e. MTcu= query field placeholder is present in the CommissioningCustomFlowUrl , the Commissioner MAY replace the placeholder value "_" in the ExpandedCommissioningCustomFlowUrl with a URL that the manufacturer custom flow can use to make a smooth return to the Commissioner when the device is in a state that it can be commissioned.

This URL field MAY contain a query component (see RFCÊ3986 section 3.4).

If a query is present, it SHALL be composed of one or more key-value pairs:

- The query SHALL use the & delimiter between key/value pairs.
- The key-value pairs SHALL follow the format name=<value> where name is the key name, and <value> is the contents of the value encoded with proper URL-encoded escaping.
- If key MTrop is present, it SHALL have a value of "_" (i.e. MTrop=_). This is the placeholder for a "returned onboarding payload" provided by the manufacturer flow to the Commissioner.
- Any key whose name begins with MT not mentioned in the previous bullets SHALL be reserved for future use by this specification.

Any other element in the CallbackUrl query field not covered by the above rules, as well as the fragment field (if present), SHALL remain as provided by the Commissioner through embedding within

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

the ExpandedCommissioningCustomFlowUrl including the order of query key/value pairs present.

5.7.3.2.1. Expansion of CallbackUrl by the manufacturer custom flow

Once the CallbackUrl is obtained by the manufacturer flow, it MAY be expanded to form a final ExpandedCallbackUrl URL to be used by proceeding with the following substitution algorithm on the provided CallbackUrl :

¥ If key MTrop is present, the manufacturer custom flow having received the initial query contain! ing the CallbackUrl MAY compute an Onboarding Payload in QR code format including MT: pre! fix, and substitute the placeholder value "_" (i.e. in MTrop=_) in the CallbackUrl with the desired contents, encoded with proper URL-encoded escaping (see RFCÊ3986 section 2).

%The contents of the MTrop=_ key/value pair in the ExpandedCallbackUrl SHALL only be expanded if the manufacturer custom flow, having received the initial query containing the CallbackUrl , supports opening a commissioning window on the target device and supports conveying the corresponding onboarding payload to the Commissioner.

%The return onboarding payload, if provided, SHALL contain an ephemeral Passcode and not a permanent code that can be used in a subsequent commissioning window. If the manufac! turer wants the Passcode embedded in the Onboarding Payload placed on-device or in pack! aging to be the one used for session establishment with the Commissioner, then the manu! facturer SHALL NOT include the MTop key in its CommissioningCustomFlowUrl and SHALL NOT populate the MTrop value in the CallbackUrl expansion.

%The contents of the return onboarding payload, if provided, SHALL be constructed to match the state of the device at the moment the ExpandedCallbackUrl is opened. At least one ingredi! ent which needs to be adapted relative to the received Onboarding Payload is the Custom Flow field which needs to be 0 for the return onboarding payload.

%The presence of this field is to assist automatically resuming commissioning without addi! tional data entry (QR code or numeric manual code) by the user at the Commissioner that initially triggered the custom flow. The manufacturer custom flow SHOULD provide an alter! nate means of conveying the onboarding payload, such as a manual pairing code.

%Note that if the information in the initial onboarding payload that caused triggering of a Custom Commissioning Flow was directly usable, it may be used by the Commissioner, either upon being triggered through the ExpandedCallbackUrl having been opened, or autonomously as a fallback.

%Commissioners providing a CallbackUrl to the manufacturer custom flow through the ExpandedCommissioningCustomFlowUrl SHOULD support using the ExpandedCallbackUrl to trig! ger resumption of Commissioning flow if the ExpandedCallbackUrl is followed, otherwise the Commissioner SHOULD NOT substitute the MTcu query field when expanding the Commission! ingCustomFlowUrl into the ExpandedCommissioningCustomFlowUrl .

%If the manufacturer custom flow failed to make the device commissionable, it SHALL NOT replace the placeholder value "_" of an included MTrop=_ key/value pair, to avoid a Commis! sioner attempting to discover or commission a device not made ready by the custom flow.

A manufacturer custom flow having received an ExpandedCommissioningCustomFlowUrl SHOULD attempt to open the ExpandedCallbackUrl , on completion of the steps, if an ExpandedCallbackUrl was computed from the CallbackUrl and opening such a URL is supported.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

5.7.3.3. Examples of CommissioningCustomFlow URLs

Below are some examples of valid ExpandedCommissioningCustomFlowUrl for several valid values of CommissioningCustomFlowUrl, as well as some examples of invalid values of CommissioningCustomFlowUrl:

- Valid URL with no query string:
  - Before expansion: https://company.example.com/matter/custom/flows/vFFF1p1234
  - After expansion: https://company.example.com/matter/custom/flows/vFFF1p1234 (no change)
- Invalid URL with no query string: http scheme is not allowed:
  - http://company.example.com/matter/custom/flows/vFFF1p1234
- Valid URL with basic manufacturer-specific scheme for query:
  - Before expansion: https://company.example.com/matter/custom/flows?vid=FFF1&pid=1234
  - After expansion: https://company.example.com/matter/custom/flows?vid=FFF1&pid=1234 (no change)
- Valid URL with MTop=_ placeholder using QR format onboarding payload embedding:
  - Before expansion: https://company.example.com/matter/custom/flows?vid=FFF1&pid=1234&MTop=_
  - After expansion: https://company.example.com/matter/custom/flows?vid=FFF1&pid=1234&MTop=MT%3A-MOA57ZU02IT2L2BJ00
    - & Onboarding payload QR content MT:-MOA57ZU02IT2L2BJ00 was embedded within MTop key
- Valid URL with MTop=_ placeholder using numeric manual code onboarding payload embedding:
  - Before expansion: https://company.example.com/matter/custom/flows?vid=FFF1&pid=1234&MTop=_
  - After expansion: https://company.example.com/matter/custom/flows?vid=FFF1&pid=1234&MTop=61040314666552104660 0
    - & Onboarding numeric manual code 61040314666552104660 0 was embedded within MTop key
- Valid URL with MTop=_ placeholder using numeric manual code onboarding payload embedding, using a different order of keys/value pairs than the previous example:
  - Before expansion: https://company.example.com/matter/custom/flows?pid=1234&MTop=_&vid=FFF1
  - After expansion: https://company.example.com/matter/custom/flows?pid=1234&MTop=61040314666552104660 0&vid=FFF1
    - & Onboarding numeric manual code 61040314666552104660 0 was embedded within MTop key
- Valid URL with onboarding payload elided (because commissioner could not provide it):
  - Before expansion: https://company.example.com/matter/custom/flows?vid=FFF1&pid=1234&MTop=_
  - After expansion: https://company.example.com/matter/custom/flows?vid=FFF1&pid=1234&MTop=_ (no change)

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

¥ Valid URL, return onboarding payload and CallbackUrl requested:

%Before expansion:

```
https://company.example.com/matter/custom/flows?vid=FFF1&pid=1234&MTop=_&MTcu=_
```

%After expansion:

```
https://company.example.com/matter/custom/flows?vid=FFF1&pid=1234&MTop=MT%3A-MOA57ZU02IT2L2BJ00&MTcu=https%3A%2F%2Fcommissioner.example.com%2Fcb%3Ftoken%3DsJ6_vqbr-vjDiG_w%253D%253D%26MTrop%3D_
```

%The ExpandedCommissioningCustomFlow URL contains:

& An embedded onboarding payload QR content value of MT:-MOA57ZU02IT2L2BJ00

& A CallbackUrl with a Commissioner-provided arbitrary token= key/value pair and the MTrop= key/value pair place-holder to indicate support for a return onboarding payload: https://commissioner.example.com/cb?token=mAsJ6_vqbr-vjDiG_w%3D%3D&MTrop=_

& After expansion of the CallbackUrl (MTcu key) into an ExpandedCallbackUrl, with an example return onboarding payload of MT:-MOA5.GB00V68T62O10, the ExpandedCallbackUrl would be:

```
https://commissioner.example.com/cb?token=mAsJ6_vqbr-vjDiG_w%3D%3D&MTrop=MT%3A-MOA5.GB00V68T62O10
```

Note that the MTcu key/value pair was initially provided URL-encoded within the ExpandedCommissioningCustomFlow URL and the MTrop=_ key/value pair placeholder now contains a substituted returned onboarding payload.

¥ Invalid URL, due to MTza=79 key/value pair in reserved MT- prefixed keys reserved for future use:

%https://company.example.com/matter/custom/flows?vid=FFF1&pid=1234&MTop=_&MTza=79

5.7.3.4. Example Custom Commissioning Flow

An example of this flow is illustrated below. The "DCL info" concept denotes that the Commissioner SHALL collect the information from the DCL via some mechanism, such as a network resource accessible to the Commissioner containing a replicated set of the DCLÕs content.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

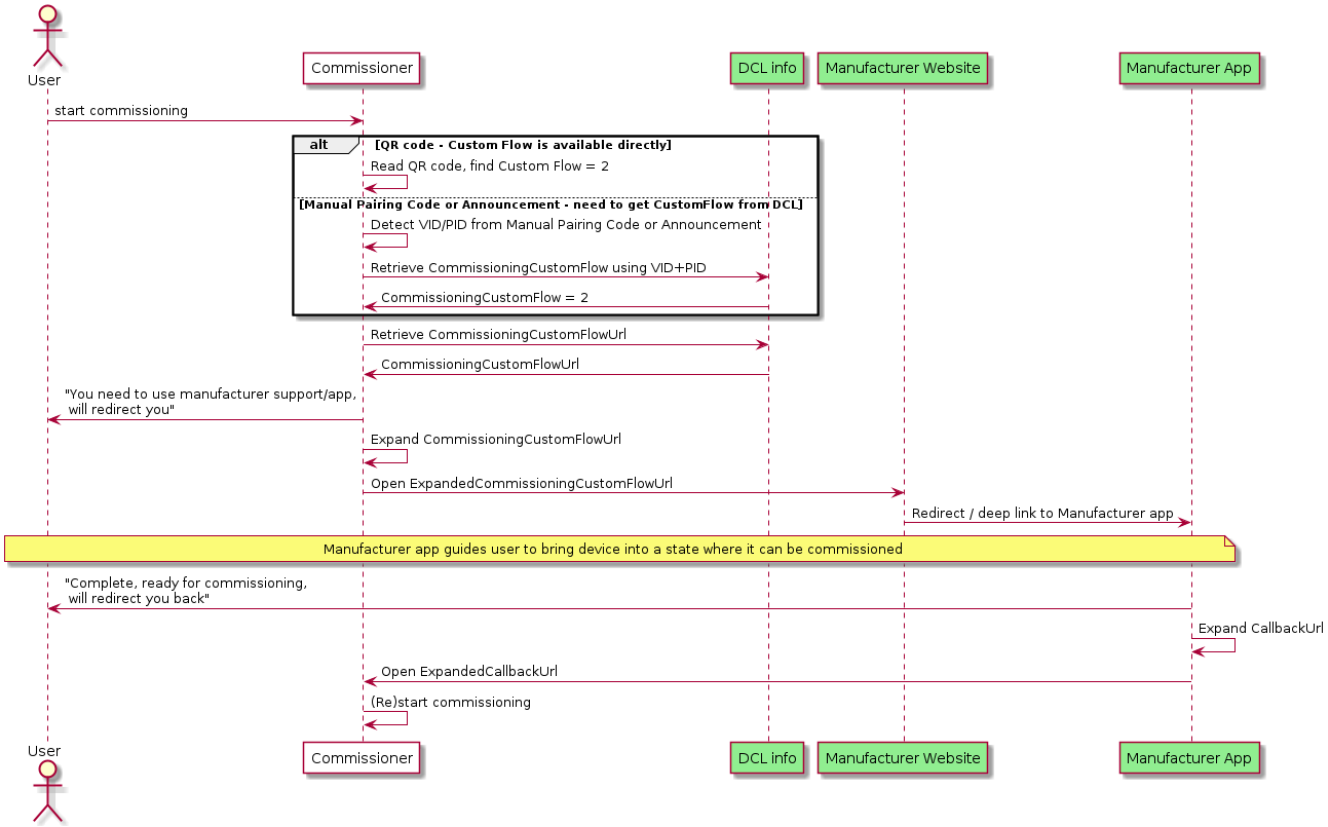


Figure 33. Custom Commissioning Flow sequence diagram

In the flow above:

- In the final steps, the User has to perform the trigger to the first Commissioner, so that it can start or continue the commissioning process.
- If possible, a Commissioner MAY continue to scan for announcements from the device in the background while any manufacturer-specific app is configuring the device to be available for commissioning. The Commissioner may need a new OnboardingPayload provided to the User by the Manufacturer Website or App.
- In order to simplify the flow, the Commissioner MAY:
  - Include the onboarding payload obtained from the user (see MTopkey in Section 5.7.3.1, ÒCommissioningCustomFlowUrl formatÓ) within the CommissioningCustomFlowUrl
  - Include a callback URL (see MTcukey in Section 5.7.3.1, ÒCommissioningCustomFlowUrl for! matÓ) within the ExpandedCommissioningCustomFlowUrl
- The Manufacturer Website or App MAY utilize the CallbackUrl field, if provided in the query string, in order to simplify the process for signaling the completion of the manufacturer-specific part of the flow back to the Commissioner. When doing so, the Manufacturer Website or App SHOULD put the device into Commissioning mode and SHOULD provide the corresponding onboarding payload to the Commissioner using the MTropkey/value pair within the Expanded! CallbackUrl .

5.7.4. Manual Pairing Code and QR Code Inclusion

Manual Pairing Code and QR setup codes enable secure commissioning and provide a consistent experience that many users are familiar with. However, because they contain a symmetric security

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

code it is not appropriate in all circumstances to have them be in a readily accessible location on the device, such as printed on the back.

The following are the requirements and recommendations regarding the QR Code and Manual Pair! ing code for Standard and User Intent Commissioning Flow Devices. Custom Commissioning Flow Device rules are described in the Custom Commissioning Flow .

The term 'on-device' allows for a physical label affixed to the device or printed directly on the device, as well as one that can be displayed on demand through some physical interface properties of the device (e.g. visual or audio).

1. Devices SHALL include the Manual Pairing code on-device or in packaging.
2. Devices SHALL NOT have the QR nor the manual pairing code in an unprotected format on the outer packaging.
3. Devices SHOULD include the QR Code, and SHOULD include it alongside the Manual Pairing Code on-device or in packaging.
4. Manual Pairing Code and QR Code on-device MAY be removable or obscured to allow the owner to prevent commissioning without their consent.
5. Devices MAY include the QR Code and Manual Pairing Code in multiple forms (see below).

Presentation of the QR Code and Manual Pairing code on-device can occur in many forms to allow for adherence to device security requirements and manufacturing considerations. For example security devices could limit the access to the QR code or Manual Pairing Code to avoid an unautho! rized user obtaining the information by simple inspection, or make the QR code and/or Manual Pairing Code removable.

The following is a list of possible ways that are acceptable to satisfy the requirements of inclusion of the QR code and Manual Pairing Code. An entry in the list should not be interpreted as being mutually exclusive with another entry. A device SHOULD include as many of these ways as possible.

¥ QR and Manual Pairing Code shown via an on-device display (when available)

¥ QR and Manual Pairing Code printed on-device, with removal/obscuring considerations noted above.

¥ Manual Pairing Code presented on-device via audio output (when available)

¥ QR and Manual Pairing Code printed on in-packaging materials.

The following are examples of QR code and Manual Pairing Code inclusion.

¥ QR Code and Manual Pairing Code printed on a Matter wireless shade inside the battery com! partment cover, and provided in the packaging.

¥ QR Code and Manual Pairing code on a Matter Smart Thermostat that can be activated via an on-device User Interface and displayed only on screen.

¥ QR Code and Manual Pairing code for a security sensor that is provided in the packaging, and on-device hidden behind a tamper-monitored cover.

¥ QR code provided on an E12 light bulb, with manual pairing code on a removable label (the area of QR code likely fits better on small form factor bulb than the area for a 13 character

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

string).

¥ A wearable device with only a Manual Pairing Code printed on the fabric. No QR code is present because of the difficulty in scanning a QR code on an irregular surface.

¥ A Smart speaker, without printed QR or manual pairing code on the device (but possibly in-packaging), that can be triggered to read out a Manual Pairing Code.

5.8. In-field Upgrade to Matter

This (informative) section discusses the case of a pre-Matter device currently in the userÕs home which gets software updated to support Matter, and which steps (either Matter-specified or manu! facturer specific) would typically be applied to accomplish this goal.

¥ The initial situation is a device which is connected to the local network, and some manufacturer specific means (e.g. a manufacturer-provided app) is used to provide new firmware (including Matter functionality) to the device, along with the associated Certification Declaration . Also, a unique Device Attestation Certificate is provided into the device using secure, manufacturer-specific means.

¥ The device restarts to enable the new firmware, and is now an uncommissioned Matter device.

¥ The device can be commissioned by any Commissioner; the Onboarding Payload needs to be provided to that Commissioner (since this information is not provided on or with the device out of the factory).

%For this, similar mechanisms as discussed as in Section 5.6.3, ÒEnhanced Commissioning Method (ECM)Ó can be employed:

& information equivalent to the parameters of the Open Commissioning Window com! mand is sent to the device using some secure manufacturer-defined means

& presentation of the passcode and other relevant information can be performed using the mechanisms described in Section 5.6.3.1, ÒPresentation of Onboarding Payload for ECMÓ .

%For devices with a means to output the Onboarding Payload themselves (e.g. device with a display or audio output), alternatively, similar mechanisms as discussed as in Section 5.6.2, ÒBasic Commissioning Method (BCM)Ó can be employed:

& information equivalent to the parameters of the Open Basic Commissioning Window command is sent to the device using some secure manufacturer-defined means

& the device itself presents Onboarding Payload .

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Chapter 6. Device Attestation and Operational Credentials

This chapter describes the procedures and cryptographic credentials involved in establishing trust between entities.

The Device Attestation section provides mechanisms for Commissioners and Administrators to determine whether a Node is a genuine certified product before sharing sensitive information such as keys and other credentials. The Device Attestation feature relies on a Device Attestation Certifi! cate (DAC) chain and on a Certification Declaration (CD) .

The Node Operational Credentials section describes the credentials used by all Nodes to mutually authenticate each other during Certificate-Authenticated Session Establishment , including the Node Operational Certificate (NOC) chain. These credentials form the basis of how Nodes are identified and take part in securing operational unicast communication .

6.1. Common Conventions

This chapter makes use of digital certificates in several subsections. All certificates within this spec! ification are based on X.509v3-compliant certificates as defined in RFCÊ5280The storage format of the certificates depends on application (e.g., DAC or NOC chain), but all certificates are directly com! patible with X.509v3 DER representation after suitable loading or decompression.

In order to simplify further exposition, this subsection contains some common normative conven! tions that SHALL apply to all digital certificates described in this specification.

The following certificate formats are defined within this specification:

¥ Compressed Node Operational credentials certificate chain elements in Matter Operational Cer! tificate Encoding or "Matter Certificate" format:

%Node Operational Certificate (NOC)

%Intermediate CA Certificate (ICAC)

%Root CA Certificate (RCAC)

¥ Device Attestation certificate chain elements in Standard X.509 DER format:

%Device Attestation Certificate (DAC): see Section 6.2.2.3, ÒDevice Attestation Certificate (DAC)Ó

%Product Attestation Intermediate (PAI): see Section 6.2.2.4, ÒProduct Attestation Intermediate (PAI) CertificateÓ

%Product Attestation Authority (PAA): see Section 6.2.2.5, ÒProduct Attestation Authority (PAA) CertificateÓ

6.1.1. Encoding of Matter-specific RDNs

In addition to the standard DN (Distinguished Names) attribute types that appear in certificate Sub! ject and Issuer fields, there are Matter-specific DN attribute types under the 1.3.6.1.4.1.37244 pri!

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

vate arc. These are listed in Table 54, ÒMatter-specific DN Object IdentifiersÓ . These OID values are assigned by the Connectivity Standards Alliance for use with Matter. All of these Matter-specific RDNs encode values normatively defined as scalars.

When used in Matter Operational Certificate (TLV) format (see Section 6.5, ÒOperational Certificate EncodingÓ), Matter-specific DN attribute types SHALL be encoded in Matter TLV as unsigned inte! gers with the specified length.

When used in X.509 ASN.1 DER format certificate encoding, Matter-specific DN attribute types SHALL have their value encoded as either a UTF8String or PrintableString according to the table below. The values SHALL be encoded in network byte order as exactly twice their specified maxi! mum octet length, encoded as uppercase hexadecimal number format without any separators or prefix, and without omitting any leading zeroes.

For example:

¥ A scalar value 0x0123_4567_89AB_CDEF for matter-node-id :

%Scalar maximal length: 8 octets (64 bits)

%Resulting string: "0123456789ABCDEF" (without quotes)

%Resulting length: 16 characters

¥ A scalar value 0xAA_33CC for matter-noc-cat :

%Scalar maximal length: 4 octets (32 bits)

%Resulting string: "00AA33CC" (without quotes)

%Resulting length: 8 characters

Table 54. Matter-specific DN Object Identifiers

TLV Tag	Matter name	Length (octets)	String length	ASN.1 OID	Types Allowed in X.509
17	matter-node-id	8	16	1.3.6.1.4.1.37244.1.1	UTF8String
18	matter-firmware-sign! ing-id	8	16	1.3.6.1.4.1.37244.1.2	UTF8String
19	matter-icac-id	8	16	1.3.6.1.4.1.37244.1.3	UTF8String
20	matter-rcac-id	8	16	1.3.6.1.4.1.37244.1.4	UTF8String
21	matter-fabric-id	8	16	1.3.6.1.4.1.37244.1.5	UTF8String
22	matter-noc-cat	4	8	1.3.6.1.4.1.37244.1.6	UTF8String
N/A	matter-oid-vid	2	4	1.3.6.1.4.1.37244.2.1	UTF8String, PrintableString

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

TLV Tag	Matter name	Length (octets)	String length	ASN.1 OID	Types Allowed in X.509
N/A	matter-oid-pid	2	4	1.3.6.1.4.1.37244.2.2	UTF8String, PrintableString

6.1.2. Key Identifier Extension Constraints

Whenever an X.509 certificate contains Authority Key Identifier or Subject Key Identifier extensions, the associated Key Identifier SHALL be of a length of 20 octets, consistent with the length of derivation method (1) described in section 4.2.1.2 of [RFC5280].

Further constraints related to the exact derivation appear in the following subsections:

- Matter Certificates (NOC, ICAC, RCAC) Subject Key Identifier extension: see Section 6.5.11.4, "Subject Key Identifier Extension"
- Matter Certificates Authority Key Identifier extension: see Section 6.5.11.5, "Authority Key Identifier Extension"
- Device Attestation Certificate (DAC) extensions: see Section 6.2.2.3, "Device Attestation Certificate (DAC)"
- Product Attestation Intermediate (PAI) Certificate extensions: see Section 6.2.2.4, "Product Attestation Intermediate (PAI) Certificate"
- Product Attestation Authority (PAA) Certificate extensions: see Section 6.2.2.5, "Product Attestation Authority (PAA) Certificate"

6.1.3. Certificate Sizes

All certificates SHALL NOT be longer than 600 bytes in their uncompressed DER format. This constraints SHALL apply to the entire DAC chain (DAC, PAI, PAA) and NOC chain (NOC, ICAC, RCAC).

Wherever Matter Operational Certificate Encoding representation is used, all certificates SHALL NOT be longer than 400 bytes in their TLV form. This constraint only applies to the NOC chain (NOC, ICAC, RCAC) since the DAC chain (DAC, PAI, PAA) only appears in DER format.

All certificates used within Matter SHOULD be as short as possible.

6.1.4. Presentation of example certificates

Certificate bodies are presented for exemplary purposes in multiple formats within this chapter. Since the translation of an X.509 certificate from ASN.1 DER format to human-readable text format may lose fidelity, especially with regards to equivalent types (e.g., PrintableString versus IA5String versus UTF8String) or serialization when non-standard OIDs are seen, textual examples SHALL NOT be considered to be normative. Only direct encoding of DER encoding, such as PEM blocks, should be used to further study the examples. In case of unforeseen divergence between an example certificate illustration and the normative rules expressed in prose, the normative prose SHALL take precedence over an ambiguous interpretation of an example.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

6.2. Device Attestation

6.2.1. Introduction

Certification of a Device includes configuring the Device with immutable credentials that can be cryptographically verified. Device Attestation is the step of the Commissioning process whereby a Commissioner cryptographically verifies a Commissionee is in fact a certified Device. This chapter describes the Device Attestation Certificate (DAC) and the systems involved in the verification of a DAC.

The processes used to convey the DAC from a Commissionee to a Commissioner, how to verify that a Commissionee holds the private key corresponding to its DAC, and specifically how the DAC is ver! ified are described in Section 6.2.3, ÒDevice Attestation ProcedureÓ .

This chapter refers to the signature algorithm ECDSA with SHA256 and to the elliptic curve secp256r1 (aka prime256v1 and NIST P-256) in compliance with the mapping for version 1.0 of the Matter Mes! sage Format of the cryptographic primitives as specified in Chapter 3, Cryptographic Primitives . Future versions of this specification might adapt these references accordingly.

6.2.2. Device Attestation Certificate (DAC)

All commissionable Matter Nodes SHALL include a Device Attestation Certificate (DAC) and corre! sponding private key, unique to that Device. The DAC is used in the Device Attestation process, as part of Commissioning a Commissionee into a Fabric. The DAC SHALL be a DER-encoded X.509v3-compliant certificate as defined in RFCÊ5280 and SHALL be issued by a Product Attestation Interme! diate (PAI) that chains directly to an approved Product Attestation Authority (PAA), and therefore SHALL have a certification path length of 2.

The DAC also SHALL contain specific values of Vendor ID and Product ID (see Section 6.2.2.2, ÒEncoding of Vendor ID and Product ID in subject and issuer fieldsÓ) in its subject field to indicate the Vendor ID and Product ID provenance of the attestation certificate. See Section 6.2.3.1, ÒAttesta! tion Information ValidationÓ for how these are used.

The validity period of a DAC is determined by the vendor and MAY be set to the maximum allowed value of 99991231235959Z GeneralizedTime to indicate that the DAC has no well-defined expiration date.

The notation used in this section to describe the specifics of the DAC uses the ASN.1 basic notation as defined in X.680. The notation below also leverages types defined in RFCÊ5280 such as Algorith! mIdentifier , RelativeDistinguishedName , Validity , Time, UTCTime, GeneralizedTime, or permitted exten! sion types. Additionally, the notation below uses the ASN.1 definitions captured in the figure below:

```
-- Matter signatures are ECDSA with SHA256

MatterSignatureIdentifier ::= SEQUENCE {
Ê   algorithm OBJECT IDENTIFIER(id-x962-ecdsa-with-sha256) }

-- Matter Names
```

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

```
-- The second to last RelativeDistinguishedName object in MatterDACName SEQUENCE SHALL contain an
-- attribute with type equal to matter-oid-vid and the last RelativeDistinguishedName object in the
-- SEQUENCE SHALL contain an attribute with type field set to matter-oid-pid

MatterDACName ::= SEQUENCE OF RelativeDistinguishedName

-- There are two acceptable formats for MatterPA name. The first is identical to MatterDACName,
-- i.e. the second to last RelativeDistinguishedName object in MatterPAName SEQUENCE SHALL contain an
-- attribute with type equal to matter-oid-vid and the last RelativeDistinguishedName object in the
-- SEQUENCE SHALL contain an attribute with type field set to matter-oid-pid. In the second acceptable
-- format, the last element of the MatterPAName SEQUENCE SHALL be an RelativeDistinguishedName with
-- an attribute with type field set to matter-oid-vid

MatterPAName ::= SEQUENCE OF RelativeDistinguishedName

-- Object definitions and references

-- X962 OIDs

id-x962 OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) ansi-x962(10045) }

id-x962-ecdsa-with-sha256 OBJECT IDENTIFIER ::= { id-x962 signatures(4) ecdsa-with-SHA2(3) ecdsa-with-SHA256(2) }

id-x962-prime256v1 OBJECT IDENTIFIER ::= { id-x962 curves(3) prime(1) prime256v1(7) }

-- CSA and Matter specific OIDs

csa-root OBJECT IDENTIFIER ::= { iso(1) identified-organization(3) dod(6) internet(1) private(4) enterprise(1) zigbee(37244) }

-- root arc for attestation certificates
matter-att-root OBJECT IDENTIFIER ::= { csa-root 2 }

-- Matter Device Attestation Certificate DN attribute for the Vendor ID (VID)
matter-oid-vid OBJECT IDENTIFIER ::= { matter-att-root 1 }

-- Matter Device Attestation Certificate DN attribute for the Product ID (PID)
matter-oid-pid OBJECT IDENTIFIER ::= { matter-att-root 2 }
```

6.2.2.1. Device Attestation Public Key Infrastructure (PKI)

The Device Attestation PKI hierarchy consists of the PAA, PAI and individual DAC. The public key

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

from the associated PAI certificate is used to cryptographically verify the DAC signature. The PAI certificate in turn is signed and attested to by the Product Attestation Authority (PAA) CA. The pub! lic key from the associated PAA certificate is used to cryptographically verify the PAI certificate sig! nature. The PAA certificate is an implicitly trusted self-signed root certificate. In this way, the DAC chains up to the PAI certificate, which in turn chains up to the PAA root certificate. A PAI SHALL be assigned to a Vendor ID value. A PAI MAY further be scoped to a single ProductID value. If a PAI is used for multiple products, then it cannot be scoped to a ProductID value, otherwise the Device Attestation Procedure will fail policy validations.

Commissioners SHALL use PAA and PAI certificates to verify the authenticity of a Commissionee before proceeding with the rest of the Commissioning flow.

The subject of all DAC and PAI certificates SHALL be unique among all those issued by their issuer, as intended by RFCÊ5280section 4.1.2.6, through the use of RelativeDistinguishedName s that ensure the uniqueness, such as for example a unique combination of commonName(OID 2.5.4.3), serialNumber (OID 2.5.4.5), organizationalUnitName (OID 2.5.4.11), etc. The exact additional constraints, including for the subject field, for PAA, PAI and DAC certificates, are presented in the following subsections.

The following figure shows the Device Attestation PKI hierarchy.

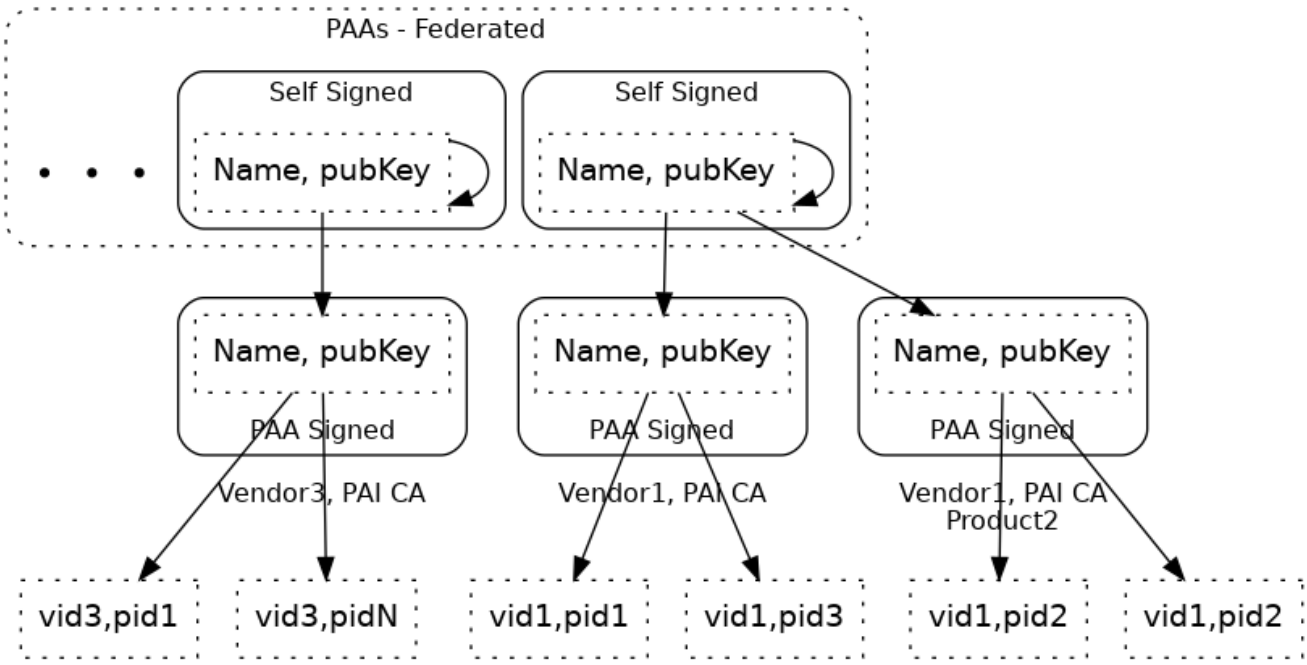


Figure 34. Device Attestation PKI hierarchy

6.2.2.2. Encoding of Vendor ID and Product ID in subject and issuer fields

The following subsections contain references to VendorID and ProductID :

- ¥ Section 6.2.2.3, ÒDevice Attestation Certificate (DAC)Ó
- ¥ Section 6.2.2.4, ÒProduct Attestation Intermediate (PAI) CertificateÓ
- ¥ Section 6.2.2.5, ÒProduct Attestation Authority (PAA) CertificateÓ

The values for VendorID and ProductID, where possible or required in issuer or subject fields SHALL be encoded by only one of two methods, without mixing the methods within a given field:

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

1. The "preferred method", using Matter-specific RelativeDistinguishedName attributes:
   a. VendorID encoded as AttributeTypeAndValue entry with type equal to 1.3.6.1.4.1.37244.2.1, and value respecting encoding specified in Section 6.1.1, ÒEncoding of Matter-specific RDNsÓ .
   b. ProductID encoded as AttributeTypeAndValue entry with type equal to 1.3.6.1.4.1.37244.2.2, and value respecting encoding specified in Section 6.1.1, ÒEncoding of Matter-specific RDNsÓ .
2. A "fallback method" to support certificate authorities that only allow customary RFCÊ5280 OIDs in the arc {joint-iso-itu-t(2) ds(5) attributeType(4)} for type values in AttributeTypeAndValue entries of RelativeDistinguishedName elements is to encode them as substrings within the commonName attribute type ({joint-iso-itu-t(2) ds(5) attributeType(4) commonName(3)}):
   a. VendorID value encoded with substring Mvid: followed by exactly 4 uppercase hexadecimal characters without elision of leading zeroes, anywhere within the commonName, such as for example:
      i. VendorID 0xFFF1 (65521 decimal): Mvid:FFF1
      ii. VendorID 0x2A (42 decimal): Mvid:002A
   b. ProductID value encoded with substring Mpid: followed by exactly 4 uppercase hexadecimal characters without elision of leading zeroes, anywhere within the commonName, such as for example:
      i. ProductID 0xC20A (49674 decimal): Mpid:C20A
      ii. ProductID 0x3A5 (933 decimal): Mpid:03A5

The "preferred method" leaves more space for content in the commonName attribute type if present. It is also less ambiguous which may allow simpler processing of certificate issuance policy validations in CAs that support the Matter-specific RelativeDistinguishedName attributes, and simplify the audit of certificates where Vendor ID and Product ID appear.

The "fallback method" is present to support less flexible CA infrastructure.

Fallback method to encode VendorID and ProductID

The "fallback method" requires exactly 9 characters that are safe to use in both PrintableString and UTF8String for either VendorID or ProductID encoding. Since these VendorID and ProductID sub-strings have unambiguous format, they MAY be provided anywhere within a commonName value, and therefore separator selection does not need to be considered. Note that the standard RFCÊ5280 length limitation for commonName attribute value is 64 characters in total (see ub-common-name in RFCÊ5280).

Using the "fallback method" for embedding of VendorID and ProductID in commonName in the subject field of a Device Attestation Certificate claiming VendorID 0xFFF1 and ProductID 0x00B1 can be illustrated with the following valid and invalid examples (without the double quotes):

- "ACME Matter Devel DAC 5CDA9899 Mvid:FFF1 Mpid:00B1" - valid and recommended since easily human-readable
- "ACME Matter Devel DAC 5CDA9899 Mpid:00B1 Mvid:FFF1" - valid and recommended since easily human-readable
- "Mpid:00B1,ACME Matter Devel DAC 5CDA9899,Mvid:FFF1" - valid example showing that order or

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

separators are not considered at all for the overall validity of the embedded fields

- "ACME Matter Devel DAC 5CDA9899 Mvid:FFF1Mpid:00B1": valid, but less readable
- "Mvid:FFF1ACME Matter Devel DAC 5CDAMpid:00B19899": valid, but highly discouraged, since embedding of substrings within other substrings may be confusing to human readers.
- "ACME Matter Devel DAC 5CDA9899 Mvid:FF1 Mpid:00B1": invalid, since substring following Mvid: is not exactly 4 uppercase hexadecimal digits
- "ACME Matter Devel DAC 5CDA9899 Mvid:fff1 Mpid:00B1": invalid, since substring following Mvid: is not exactly 4 uppercase hexadecimal digits
- "ACME Matter Devel DAC 5CDA9899 Mvid:FFF1 Mpid:B1": invalid, since substring following Mpid: is not exactly 4 uppercase hexadecimal digits
- "ACME Matter Devel DAC 5CDA9899 Mpid: Mvid:FFF1": invalid, since substring following Mpid: is not exactly 4 uppercase hexadecimal digits

If either the Vendor ID (1.3.6.1.4.1.37244.2.1) or Product ID (1.3.6.1.4.1.37244.2.2) Matter-specific OIDs appear in any RelativeDistinguishedName in the subject or issuer fields of a certificate which is part of the Device Attestation Certificate chain path, then that certificate within the chain SHALL NOT have its commonName , if present, parsed for the "fallback method", in the rest of the issuer or subject field where the Matter-specific OIDs appear. In other words, considering a field such as subject or issuer , the presence of either of these OIDs as the type for any AttributeTypeAndValue within any RelativeDistinguishedName of that field SHALL cause the "fallback method" to be skipped altogether for that field. Otherwise, when the "fallback method" can legally be used, it SHALL only be used against AttributeTypeAndValue sequences where the type field is commonName ({joint-iso-itu-t(2) ds(5) attributeType(4) commonName(3)}) in the issuer and subject fields, and any mention thereafter of using or matching a "Vendor ID" or "Product ID" with regards to a Device Attestation Procedure step SHALL rely on values obtained with that method.

For example, if a given Product Attestation Intermediate certificate has a subject field employing a particular method of encoding the VendorID and ProductID, either using only Matter-specific OIDs or only the fallback method, then it follows that a Device Attestation Certificates issued by the certificate authority of that Product Attestation Intermediate SHALL have the same DistinguishedName content in its issuer field, so that the basic path validation algorithm works. That Device Attestation Certificate MAY however have the "fallback method" used within its subject field, if the Product Attestation Intermediate certificate authority is unable to encode/reflect the Matter-specific OIDs in RelativeDistinguishedName attributes within the subject field. The rules for whether to consider the canonical or "fallback method" for VendorID and ProductID encoding applies field by field independently for each instance of subject or issuer field found in certificates within the DAC chain.

6.2.2.3. Device Attestation Certificate (DAC)

The attributes in a DAC include:

```
Certificate ::= SEQUENCE {
Ê  tbsCertificate      DACTBSCertificate,
Ê  signatureAlgorithm  AlgorithmIdentifier,
```


Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

```
Ê   signatureValue       BIT STRING }
```

```
DACTBSCertificate ::= SEQUENCE {
Ê   version               INTEGER ( v3(2) ),
Ê   serialNumber          INTEGER,
Ê   signature             MatterSignatureIdentifier,
Ê   issuer                MatterPAName,
Ê   validity              Validity,
Ê   subject               MatterDACName,
Ê   subjectPublicKeyInfo    SEQUENCE {
Ê      algorithm            OBJECT IDENTIFIER(id-x962-prime256v1),
Ê      subjectPublicKey       BIT STRING },
Ê   extensions            DACExtensions }

DACExtensions ::= SEQUENCE {
Ê  basicConstraint Extension({extnID id-ce-basicConstraints, critical TRUE, extnValue
BasicConstraints  {cA FALSE} }),
Ê  keyUsage Extension({extnID id-ce-keyUsage, critical TRUE, extnValue
KeyUsage({digitalSignature})}),
Ê  authorityKeyIdentifier Extension({extnID id-ce-authorityKeyIdentifier}),
Ê  subjectKeyIdentifier Extension({extnID id-ce-subjectKeyIdentifier}),
Ê  extendedKeyUsage Extension({extnID id-ce-extKeyUsage}) OPTIONAL,
Ê  authorityInformationAccess Extension({extnID id-pe-authorityInfoAccess}) OPTIONAL,
Ê  subjectAlternateName Extension({extnID id-ce-subjectAltName}) OPTIONAL
}
```

The DAC certificate SHALL follow the following constraints layered on top of the encoding specified by RFCÊ5280within the TBSCertificate structure:

1. The version field SHALL be set to 2 to indicate v3 certificate.
2. The signature field SHALL contain the identifier for signatureAlgorithm ecdsa-with-SHA256
3. The issuer field SHALL be a sequence of RelativeDistinguishedName s.
4. The issuer field SHALL have exactly one VendorID value present.
5. The issuer field SHALL have exactly zero or one ProductID value present.
6. The subject field SHALL be a sequence of RelativeDistinguishedName s.
7. The subject field SHALL have exactly one VendorID value present.
    a. The VendorID value present in the issuer field SHALL match the VendorID value found in subject field.
8. The subject field SHALL have exactly one ProductID value present.
    a. If a ProductID value was present in the issuer field, the ProductID value found in subject field SHALL match the value found in the issuer field.
9. The algorithm field in subjectPublicKeyInfo field SHALL be the object identifier for prime256v1.
10. The certificate SHALL carry the following Extensions:

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

a. Basic Constraint extension SHALL be marked critical and have the cA field set to FALSE.
b. Key Usage extension SHALL be marked critical
   i. The KeyUsage bitstring SHALL only have the digitalSignature bit set.
   ii. Other bits SHALL NOT be set
c. Authority Key Identifier
d. Subject Key Identifier

11. The certificate MAY also carry the following additional Extensions:
    a. Extended Key Usage
    b. Authority Information Access
    c. Subject Alternate Name

Valid example DAC with associated private key, in X.509 PEM format

```
-----BEGIN CERTIFICATE-----
MIIB6TCCAY+gAwIBAgIIDgY7dCvPvl0wCgYIKoZIzj0EAwIwRjEYMBYGA1UEAwwP
TWF0dGVyIFRlc3QgUEFJMRQwEgYKKwYBBAGConwCAQwERkZGMTEUMBIGCisGAQQB
gqJ8AgIMBDgwMDAwIBcNMjEwNjI4MTQyMzQzWhgPOTk5OTEyMzEyMzU5NTlaMEsx
HTAbBgNVBAMMFE1hdHRlciBUZXN0IERBQyAwMDAxMRQwEgYKKwYBBAGConwCAQwE
RkZGMTEUMBIGCisGAQQBgqJ8AgIMBDgwMDAwWTATBgcqhkjOPQIBBggqhkjOPQMB
BwNCAATCJYMix9xyc3wzvu1wczeqJIW8Rnk+TVrJp1rXQ1JmyQoCjuyvJlD+cAnv
/K7L6tHyw9EkNd7C6tPZkpW/ztbDo2AwXjAMBgNVHRMBAf8EAjAAMA4GA1UdDwEB
/wQEAwIHgDAdBgNVHQ4EFgQUlsLZJJTql4XA0WcI44jxwJHqD9UwHwYDVR0jBBgw
FoAUr0K3CU3r1RXsbs8zuBEVIl8yUogwCgYIKoZIzj0EAwIDSAAwRQIgX8sppA08
NabozmBlxtCdphc9xbJF7DIEkePTSTK3PhcCIQC0VpkPUgUQBFo4j3VOdxVAoESX
kjGWRV5EDWgl2WEDZA==
-----END CERTIFICATE-----

-----BEGIN EC PRIVATE KEY-----
MHcCAQEEIHtcWp+0aVVH+DAQ38iXpphqmT7LfMnMD4V/kIqszwfuoAoGCCqGSM49
AwEHoUQDQgAEwiWDIsfccnN8M77tcHM3qiSFvEZ5Pk1ayada10NSZskKAo7sryZQ
/nAJ7/yuy+rR8sPRJDXewurT2ZKVv87Www==
-----END EC PRIVATE KEY-----
```

Human-readable contents of example DAC X.509 certificate

```
Certificate:
Ê   Data:
Ê       Version: 3 (0x2)
Ê       Serial Number: 1010560536528535133 (0xe063b742bcfbe5d)
Ê       Signature Algorithm: ecdsa-with-SHA256
Ê       Issuer: CN = Matter Test PAI, 1.3.6.1.4.1.37244.2.1 = FFF1,
1.3.6.1.4.1.37244.2.2 = 8000
Ê       Validity
Ê           Not Before: Jun 28 14:23:43 2021 GMT
Ê           Not After : Dec 31 23:59:59 9999 GMT
Ê       Subject: CN = Matter Test DAC 0001, 1.3.6.1.4.1.37244.2.1 = FFF1,
```


Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

```
1.3.6.1.4.1.37244.2.2 = 8000
Ê      Subject Public Key Info:
Ê          Public Key Algorithm: id-ecPublicKey
Ê              Public-Key: (256 bit)
Ê              pub:
Ê                  04:c2:25:83:22:c7:dc:72:73:7c:33:be:ed:70:73:
Ê                  37:aa:24:85:bc:46:79:3e:4d:5a:c9:a7:5a:d7:43:
Ê                  52:66:c9:0a:02:8e:ec:af:26:50:fe:70:09:ef:fc:
Ê                  ae:cb:ea:d1:f2:c3:d1:24:35:de:c2:ea:d3:d9:92:
Ê                  95:bf:ce:d6:c3
Ê              ASN1 OID: prime256v1
Ê              NIST CURVE: P-256
Ê      X509v3 extensions:
Ê          X509v3 Basic Constraints: critical
Ê              CA:FALSE
Ê          X509v3 Key Usage: critical
Ê              Digital Signature
Ê          X509v3 Subject Key Identifier:
Ê              96:C2:D9:24:94:EA:97:85:C0:D1:67:08:E3:88:F1:C0:91:EA:0F:D5
Ê          X509v3 Authority Key Identifier:
Ê              keyid:AF:42:B7:09:4D:EB:D5:15:EC:6E:CF:33:B8:11:15:22:5F:32:52:88

Ê   Signature Algorithm: ecdsa-with-SHA256
Ê       30:45:02:20:5f:cb:29:a4:0d:3c:35:a6:e8:ce:60:65:c6:d0:
Ê       9d:a6:17:3d:c5:b2:45:ec:32:04:91:e3:d3:49:32:b7:3e:17:
Ê       02:21:00:b4:56:99:0f:52:05:10:04:5a:38:8f:75:4e:77:15:
Ê       40:a0:44:97:92:31:96:45:5e:44:0d:68:25:d9:61:03:64
```

Valid example DAC with associated private key, in X.509 PEM format, using "fallback method" for VendorID and ProductID in Subject

```
-----BEGIN CERTIFICATE-----
MIIB0DCCAXegAwIBAgIIbec9lw3wZpAwCgYIKoZIzj0EAwIwRjEYMBYGA1UEAwwP
TWF0dGVyIFRlc3QgUEFJMRQwEgYKKwYBBAGConwCAQwERkZGMTEUMBIGCisGAQQB
gqJ8AgIMBDgwMDAwIBcNMjEwNjI4MTQyMzQzWhgPOTk5OTEyMzEyMzU5NTlaMDMx
MTAvBgNVBAMMKE1hdHRlciBUZXN0IERBQyAwMDAxIE12aWQ6RkZGMSBNcGlkOjgw
MDAwWTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAATCJYMix9xyc3wzvu1wczeqJlW8
Rnk+TVrJp1rXQ1JmyQoCjuyvJlD+cAnv/K7L6tHyw9EkNd7C6tPZkpW/ztbDo2Aw
XjAMBgNVHRMBAf8EAjAAMA4GA1UdDwEB/wQEAwIHgDAdBgNVHQ4EFgQUlsLZJJTq
l4XA0WcI44jxwJHqD9UwHwYDVR0jBBgwFoAUr0K3CU3r1RXsbs8zuBEVIl8yUogw
CgYIKoZIzj0EAwIDRwAwRAIgbvYsHaGRTg1JzPTB6TqfVFPABF8LCYkEP1AvV7Ah
yL4CIACKW3A6YixqtqKfkwuvw81mMVymqafU8kx5k1c0zqbe
-----END CERTIFICATE-----
```

Human-readable contents of example DAC X.509 certificate, using "fallback method" for VendorID and Pro! ductID in Subject

```
Certificate:
Ê   Data:
Ê       Version: 3 (0x2)
```


Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

```
Ê      Serial Number: 7919366188737521296 (0x6de73d970df06690)
Ê      Signature Algorithm: ecdsa-with-SHA256
Ê      Issuer: CN = Matter Test PAI, 1.3.6.1.4.1.37244.2.1 = FFF1,
1.3.6.1.4.1.37244.2.2 = 8000
Ê      Validity
Ê         Not Before: Jun 28 14:23:43 2021 GMT
Ê         Not After : Dec 31 23:59:59 9999 GMT
Ê      Subject: CN = Matter Test DAC 0001 Mvid:FFF1 Mpid:8000
Ê      Subject Public Key Info:
Ê         Public Key Algorithm: id-ecPublicKey
Ê            Public-Key: (256 bit)
Ê            pub:
Ê               04:c2:25:83:22:c7:dc:72:73:7c:33:be:ed:70:73:
Ê               37:aa:24:85:bc:46:79:3e:4d:5a:c9:a7:5a:d7:43:
Ê               52:66:c9:0a:02:8e:ec:af:26:50:fe:70:09:ef:fc:
Ê               ae:cb:ea:d1:f2:c3:d1:24:35:de:c2:ea:d3:d9:92:
Ê               95:bf:ce:d6:c3
Ê            ASN1 OID: prime256v1
Ê            NIST CURVE: P-256
Ê      X509v3 extensions:
Ê         X509v3 Basic Constraints: critical
Ê            CA:FALSE
Ê         X509v3 Key Usage: critical
Ê            Digital Signature
Ê         X509v3 Subject Key Identifier:
Ê            96:C2:D9:24:94:EA:97:85:C0:D1:67:08:E3:88:F1:C0:91:EA:0F:D5
Ê         X509v3 Authority Key Identifier:
Ê            keyid:AF:42:B7:09:4D:EB:D5:15:EC:6E:CF:33:B8:11:15:22:5F:32:52:88

Ê   Signature Algorithm: ecdsa-with-SHA256
Ê      30:44:02:20:6e:f6:2c:1d:a1:91:4e:0d:49:cc:f4:c1:e9:3a:
Ê      9f:54:53:c0:04:5f:0b:09:89:04:3f:50:2f:57:b0:21:c8:be:
Ê      02:20:00:8a:5b:70:3a:62:2c:6a:b6:a2:9f:93:0b:af:c3:cd:
Ê      66:31:5c:a6:a9:a7:d4:f2:4c:79:93:57:34:ce:a6:de
```

6.2.2.4. Product Attestation Intermediate (PAI) Certificate

The attributes in a PAI certificate include:

```
Certificate ::= SEQUENCE {
Ê   tbsCertificate       PAITBSCertificate,
Ê   signatureAlgorithm   AlgorithmIdentifier,
Ê   signatureValue       BIT STRING }
```

```
PAITBSCertificate ::= SEQUENCE {
Ê   version              INTEGER ( v3(2) ),
Ê   serialNumber           INTEGER,
Ê   signature            MatterSignatureIdentifier,
```


Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

```
Ê  issuer                Name,
Ê  validity              Validity,
Ê  subject               MatterPAName,
Ê  subjectPublicKeyInfo  SEQUENCE {
Ê    algorithm           OBJECT IDENTIFIER(id-x962-prime256v1),
Ê    subjectPublicKey    BIT STRING },
Ê  extensions            PAIExtensions }

PAIExtensions ::= SEQUENCE {
Ê  basicConstraint Extension({extnID id-ce-basicConstraints, critical TRUE, extnValue
BasicConstraints  {cA TRUE, pathLen 0} }),
Ê  keyUsage Extension({extnID id-ce-keyUsage, critical TRUE, extnValue KeyUsage(<see
text>)}),
Ê  authorityKeyIdentifier Extension({extnID id-ce-authorityKeyIdentifier}),
Ê  subjectKeyIdentifier Extension({extnID id-ce-subjectKeyIdentifier}),
Ê  extendedKeyUsage Extension({extnID id-ce-extKeyUsage}) OPTIONAL
}
```

The PAI certificate SHALL follow the following constraints layered on top of the encoding specified by RFCÊ5280within the TBSCertificate structure:

1. The version field SHALL be set to 2 to indicate v3 certificate.
2. The signature field SHALL contain the identifier for signatureAlgorithm ecdsa-with-SHA256
3. The issuer field SHALL be a sequence of RelativeDistinguishedName s.
4. The issuer field SHALL have exactly zero or one VendorID value present.
5. The subject field SHALL be a sequence of RelativeDistinguishedName s.
6. The subject field SHALL have exactly one VendorID value present.
    a. If a VendorID value was present in the issuer field, the VendorID value found in subject field SHALL match the value found in the issuer field.
7. The subject field SHALL have exactly zero or one ProductID value present.
8. The algorithm field in subjectPublicKeyInfo field SHALL be the object identifier for prime256v1.
9. The certificate SHALL carry the following Extensions:
    a. Basic Constraint extension SHALL be marked critical and have the cA field set to TRUE and pathLen field set to 0.
    b. Key Usage extension SHALL be marked critical
        i. Both the keyCertSign and cRLSignbits SHALL be set in the KeyUsagebitstring
        ii. The digitalSignature bit MAY be set in the KeyUsagebitstring
        iii. Other bits SHALL NOT be set
    c. Authority Key Identifier
    d. Subject Key Identifier
10. The certificate MAY also carry the following additional Extensions:
    a. Extended Key Usage

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

The PAI certificate presented in the following example is for the issuer of the example DAC certifi! cate from the previous section.

Valid example PAI with associated private key, in X.509 PEM format

```
-----BEGIN CERTIFICATE-----
MIIB1DCCAXqgAwIBAgIIPmzmUJrYQM0wCgYIKoZIzj0EAwIwMDEYMBYGA1UEAwwP
TWF0dGVyIFRlc3QgUEFBMRQwEgYKKwYBBAGConwCAQwERkZGMTAgFw0yMTA2Mjgx
NDIzNDNaGA85OTk5MTIzMTIzNTk1OVowRjEYMBYGA1UEAwwPTWF0dGVyIFRlc3Qg
UEFJMRQwEgYKKwYBBAGConwCAQwERkZGMTEUMBIGCisGAQQBgqJ8AgIMBDgwMDAw
WTATBgcqhkjOPQIBBggqhkjOPQMBBwNCAASA3fEbIo8+MfY7z1eY2hRiOuu96C7z
eO6tv7GP4avOMdCO1LIGBLbMxtm1+rZOfeEMt0vgF8nsFRYFbXDyzQsio2YwZDAS
BgNVHRMBAf8ECDAGAQH/AgEAMA4GA1UdDwEB/wQEAwIBBjAdBgNVHQ4EFgQUr0K3
CU3r1RXsbs8zuBEVIl8yUogwHwYDVR0jBBgwFoAUav0idx9RH+y/FkGXZxDc3DGh
cX4wCgYIKoZIzj0EAwIDSAAwRQIhAJbJyM8uAYhgBdj1vHLAe3X9mldpWsSRETET
i+oDPOUDAiAlVJQ75X1T1sR199I+v8/CA2zSm6Y5PsfvrYcUq3GCGQ==
-----END CERTIFICATE-----

-----BEGIN EC PRIVATE KEY-----
MHcCAQEEIEZ7LYpps1z+a9sPw2qBp9jj5F0GLffNuCJY88hAHcMYoAoGCCqGSM49
AwEHoUQDQgAEgN3xGyKPPjH2O89XmNoUYjrrvegu83jurb+xj+GrzjHQjtSyBgS2
zMbZtfq2Tn3hDLdL4BfJ7BUWBW1w8s0LIg==
-----END EC PRIVATE KEY-----
```

Human-readable contents of example PAI X.509 certificate

```
Certificate:
Ê   Data:
Ê       Version: 3 (0x2)
Ê       Serial Number: 4498223361705918669 (0x3e6ce6509ad840cd)
Ê       Signature Algorithm: ecdsa-with-SHA256
Ê       Issuer: CN = Matter Test PAA, 1.3.6.1.4.1.37244.2.1 = FFF1
Ê       Validity
Ê           Not Before: Jun 28 14:23:43 2021 GMT
Ê           Not After : Dec 31 23:59:59 9999 GMT
Ê       Subject: CN = Matter Test PAI, 1.3.6.1.4.1.37244.2.1 = FFF1,
1.3.6.1.4.1.37244.2.2 = 8000
Ê       Subject Public Key Info:
Ê           Public Key Algorithm: id-ecPublicKey
Ê               Public-Key: (256 bit)
Ê               pub:
Ê                   04:80:dd:f1:1b:22:8f:3e:31:f6:3b:cf:57:98:da:
Ê                   14:62:3a:eb:bd:e8:2e:f3:78:ee:ad:bf:b1:8f:e1:
Ê                   ab:ce:31:d0:8e:d4:b2:06:04:b6:cc:c6:d9:b5:fa:
Ê                   b6:4e:7d:e1:0c:b7:4b:e0:17:c9:ec:15:16:05:6d:
Ê                   70:f2:cd:0b:22
Ê               ASN1 OID: prime256v1
Ê               NIST CURVE: P-256
Ê       X509v3 extensions:
Ê           X509v3 Basic Constraints: critical
```


Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

```
Ê            CA:TRUE, pathlen:0
Ê         X509v3 Key Usage: critical
Ê            Certificate Sign, CRL Sign
Ê         X509v3 Subject Key Identifier:
Ê            AF:42:B7:09:4D:EB:D5:15:EC:6E:CF:33:B8:11:15:22:5F:32:52:88
Ê         X509v3 Authority Key Identifier:
Ê            keyid:6A:FD:22:77:1F:51:1F:EC:BF:16:41:97:67:10:DC:DC:31:A1:71:7E

Ê   Signature Algorithm: ecdsa-with-SHA256
Ê       30:45:02:21:00:96:c9:c8:cf:2e:01:88:60:05:d8:f5:bc:72:
Ê       c0:7b:75:fd:9a:57:69:5a:c4:91:11:31:13:8b:ea:03:3c:e5:
Ê       03:02:20:25:54:94:3b:e5:7d:53:d6:c4:75:f7:d2:3e:bf:cf:
Ê       c2:03:6c:d2:9b:a6:39:3e:c7:ef:ad:87:14:ab:71:82:19
```

6.2.2.5. Product Attestation Authority (PAA) Certificate

The attributes in a PAA certificate include:

```
Certificate ::= SEQUENCE {
Ê   tbsCertificate       PAATBSCertificate,
Ê   signatureAlgorithm   AlgorithmIdentifier,
Ê   signatureValue       BIT STRING }
```

```
PAATBSCertificate ::= SEQUENCE {
Ê   version                INTEGER ( v3(2) ),
Ê   serialNumber           INTEGER,
Ê   signature              MatterSignatureIdentifier,
Ê   issuer                 Name,
Ê   validity               Validity,
Ê   subject                Name,
Ê   subjectPublicKeyInfo    SEQUENCE {
Ê      algorithm             OBJECT IDENTIFIER(id-x962-prime256v1),
Ê      subjectPublicKey      BIT STRING },
Ê   extensions             PAAExtensions }

PAAExtensions ::= SEQUENCE {
Ê  basicConstraint Extension({extnID id-ce-basicConstraints, critical TRUE, extnValue
BasicConstraints  {cA TRUE, pathLen 1} }),
Ê  keyUsage Extension({extnID id-ce-keyUsage, critical TRUE, extnValue KeyUsage(<see
text>)}),
Ê  authorityKeyIdentifier Extension({extnID id-ce-authorityKeyIdentifier}) OPTIONAL,
Ê  subjectKeyIdentifier Extension({extnID id-ce-subjectKeyIdentifier}),
Ê  extendedKeyUsage Extension({extnID id-ce-extKeyUsage}) OPTIONAL
}
```

The PAA certificate SHALL follow the following constraints layered on top of the encoding specified by RFCÊ5280within the TBSCertificate structure:

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

1. The version field SHALL be set to 2 to indicate v3 certificate.
2. The signature field SHALL contain the identifier for signatureAlgorithm ecdsa-with-SHA256.
3. The issuer field SHALL be a sequence of RelativeDistinguishedName s.
4. The issuer field SHALL have exactly zero or one VendorID value present.
5. The subject field SHALL be a sequence of RelativeDistinguishedName s.
6. The subject field SHALL have exactly zero or one VendorID value present.
7. The issuer and subject fields SHALL match exactly.
8. A ProductID value SHALL NOT be present in either the subject or issuer fields.
9. The algorithm field in subjectPublicKeyInfo field SHALL be the object identifier for prime256v1.
10. The certificate SHALL carry the following Extensions:
    a. Basic Constraint extension SHALL be marked critical and have the cA field set to TRUE. The 'pathLen' field MAY be set and if the 'pathLen' is field is present it SHALL be set to 1.
    b. Key Usage extension SHALL be marked critical .
        i. Both the keyCertSign and cRLSign bits SHALL be set in the KeyUsage bitstring
        ii. The digitalSignature bit MAY be set in the KeyUsage bitstring
        iii. Other bits SHALL NOT be set
    c. Subject Key Identifier
11. The certificate MAY also carry the following additional Extensions:
    a. Extended Key Usage
    b. Authority Key Identifier

The PAA certificate presented in the following example is for the issuer of the example PAI certifi! cate from the previous section.

Valid example PAA with associated private key, in X.509 PEM format

```
-----BEGIN CERTIFICATE-----
MIIBvTCCAWSgAwIBAgIITqjoMYLUHBwwCgYIKoZIzj0EAwIwMDEYMBYGA1UEAwwP
TWF0dGVyIFRlc3QgUEFBMRQwEgYKKwYBBAGConwCAQwERkZGMTAgFw0yMTA2Mjgx
NDIzNDNaGA85OTk5MTIzMTIzNTk1OVowMDEYMBYGA1UEAwwPTWF0dGVyIFRlc3Qg
UEFBMRQwEgYKKwYBBAGConwCAQwERkZGMTBZMBMGByqGSM49AgEGCCqGSM49AwEH
A0IABLbLY3KIfyko9brIGqnZOuJDHK2p154kL2UXfvnO2TKijs0Duq9qj8oYShpQ
NUKWDUU/MD8fGUIddR6Pjxqam3WjZjBkMBIGA1UdEwEB/wQIMAYBAf8CAQEwDgYD
VR0PAQH/BAQDAgEGMB0GA1UdDgQWBBRq/SJ3H1Ef7L8WQZdnENzcMaFxfjAfBgNV
HSMEGDAWgBRq/SJ3H1Ef7L8WQZdnENzcMaFxfjAKBggqhkjOPQQDAgNHADBEAiBQ
qoAC9NkyqaAFOPZTaK0P/8jvu8m+t9pWmDXPmqdRDgIgI7rI/g8j51RFtlM5CBpH
mUkpxyqvChVI1A0DTVFLJd4=
-----END CERTIFICATE-----

-----BEGIN EC PRIVATE KEY-----
MHcCAQEEIGUSyuyuz8VD1gYjFhWXFi8BRoTFZaEpti/SjCerHMxQoAoGCCqGSM49
AwEHoUQDQgAEtstjcoh/KSj1usgaqdk64kMcranXniQvZRd++c7ZMqKOzQO6r2qP
```


Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

```
yhhKGlA1QpYNRT8wPx8ZQh11Ho+PGpqbdQ==
-----END EC PRIVATE KEY-----
```

Human-readable contents of example PAA X.509 certificate

```
Certificate:
Ê   Data:
Ê      Version: 3 (0x2)
Ê      Serial Number: 5668035430391749660 (0x4ea8e83182d41c1c)
Ê      Signature Algorithm: ecdsa-with-SHA256
Ê      Issuer: CN = Matter Test PAA, 1.3.6.1.4.1.37244.2.1 = FFF1
Ê      Validity
Ê         Not Before: Jun 28 14:23:43 2021 GMT
Ê         Not After : Dec 31 23:59:59 9999 GMT
Ê      Subject: CN = Matter Test PAA, 1.3.6.1.4.1.37244.2.1 = FFF1
Ê      Subject Public Key Info:
Ê         Public Key Algorithm: id-ecPublicKey
Ê            Public-Key: (256 bit)
Ê            pub:
Ê               04:b6:cb:63:72:88:7f:29:28:f5:ba:c8:1a:a9:d9:
Ê               3a:e2:43:1c:ad:a9:d7:9e:24:2f:65:17:7e:f9:ce:
Ê               d9:32:a2:8e:cd:03:ba:af:6a:8f:ca:18:4a:1a:50:
Ê               35:42:96:0d:45:3f:30:3f:1f:19:42:1d:75:1e:8f:
Ê               8f:1a:9a:9b:75
Ê            ASN1 OID: prime256v1
Ê            NIST CURVE: P-256
Ê      X509v3 extensions:
Ê         X509v3 Basic Constraints: critical
Ê            CA:TRUE, pathlen:1
Ê         X509v3 Key Usage: critical
Ê            Certificate Sign, CRL Sign
Ê         X509v3 Subject Key Identifier:
Ê            6A:FD:22:77:1F:51:1F:EC:BF:16:41:97:67:10:DC:DC:31:A1:71:7E
Ê         X509v3 Authority Key Identifier:
Ê            keyid:6A:FD:22:77:1F:51:1F:EC:BF:16:41:97:67:10:DC:DC:31:A1:71:7E

Ê   Signature Algorithm: ecdsa-with-SHA256
Ê      30:44:02:20:50:aa:80:02:f4:d9:32:a9:a0:05:38:f6:53:68:
Ê      ad:0f:ff:c8:ef:bb:c9:be:b7:da:56:98:35:cf:9a:a7:51:0e:
Ê      02:20:23:ba:c8:fe:0f:23:e7:54:45:b6:53:39:08:1a:47:99:
Ê      49:29:c7:2a:af:0a:15:48:d4:0d:03:4d:51:4b:25:de
```

6.2.3. Device Attestation Procedure

The device attestation procedure SHALL be executed by Commissioners when commissioning a device. It serves to validate whether a particular device is certified for Matter compliance and that it was legitimately produced by the certified manufacturer. See Section 5.5, ÒCommissioning FlowsÓ for the possible outcomes based on whether the Device Attestation Procedure succeeds or fails to attest the device.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

The Device Attestation Certificate Chain MAY be read at any time, either prior to or after receipt of the AttestationResponse . The Commissionee SHALL make the Certificate Chain available whenever requested using Section 11.17.7.3, ÒCertificateChainRequest CommandÓ . If the Commissioner does not already have this information, to proceed with the validation, it SHALL request the Commis! sioneeÕs Device Attestation Certificate Chain using Section 11.17.7.3, ÒCertificateChainRequest Com! mandÓ.

The procedure is as follows:

1. The Commissioner SHALL generate a random 32 byte attestation nonce using Crypto_DRBG().
2. The Commissioner SHALL send the AttestationNonce to the Commissionee and request Attesta! tion Information using Section 11.17.7.1, ÒAttestationRequest CommandÓ .
3. The Commissionee SHALL return the signed Attestation Information to the Commissioner using Section 11.17.7.2, ÒAttestationResponse CommandÓ.

After execution of the procedure, the Attestation Information SHOULD be validated using the checks described in Section 6.2.3.1, ÒAttestation Information ValidationÓ .

6.2.3.1. Attestation Information Validation

A Commissioner validating the Attestation Information SHOULD record sufficient information to provide detailed results of the validation outcome to users. Therefore, prior to validating Attesta! tion Information, a Commissioner SHOULD have previously obtained the Device Attestation Certifi! cate chain for the Commissionee, so that the DAC and PAI necessary for the procedure are available.

In order to consider a Commissionee successfully attested, a Commissioner SHALL have success! fully validated at least the following:

- ¥ The PAA SHALL be validated for presence in the CommissionerÕs trusted root store, which SHOULD include at least the set of globally trusted PAA certificates present in the Distributed Compliance Ledger at the issuing timestamp (notBefore) of the DAC.
- ¥ The DAC certificate chain SHALL be validated using the Crypto_VerifyChainDER() function, tak! ing into account the mandatory presence of the PAI and of the PAA. It is especially important to ensure the entire chain has a length of exactly 3 elements (PAA certificate, PAI certificate, Device Attestation Certificate) and that the necessary format policies previously exposed are validated, to avoid unauthorized path chaining (e.g., through multiple PAI certificates).
  - %Chain validation SHALL be performed with respect to the notBefore timestamp of the DAC to ensure that the DAC was valid when it was issued. This way of validating is abided by the Crypto_VerifyChainDER() function.
  - %Chain validation SHALL include revocation checks of the DAC, PAI and PAA, based on the CommissionerÕs best understanding of revoked entities.
- ¥ The VendorID value found in the subject DN of the DAC SHALL match the VendorID value in the subject DN of the PAI certificate.
- ¥ If the PAA certificate contains a VendorID value in its subject DN, its value SHALL match the VendorID value in the subject DN of the PAI certificate.
- ¥ The Device Attestation Signature (attestation_signature) field from Attestation Response SHALL

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

be validated:

```
Success = Crypto_Verify(
Ê  publicKey = Public key from DAC,
Ê  message = Attestation Information TBS (  attestation_tbs  ),
Ê  signature = Device Attestation Signature (  attestation_signature  )
)
```

where the fields are encoded as described in Section 11.17.5.5, ÒAttestation InformationÓ .

%The AttestationChallenge SHALL be obtained from a CASE session, resumed CASE session, or PASE session depending on the method used to establish the secure session within which device attestation is conducted.

¥ The AttestationNonce in Device Attestation elements SHALL match the CommissionerÕs pro! vided AttestationNonce.

¥ The Certification Declaration signature SHALL be validated using the Crypto_Verify() function and the public key obtained from the CSAÕs Certificate Authority Certificate.

¥ The Certification Declaration SHALL be validated:

%The vendor_id field in the Certification Declaration SHALL match the VendorID attribute found in the Basic Information cluster.

%The product_id_array field in the Certification Declaration SHALL contain the value of the ProductID attribute found in the Basic Information cluster.

%The Certification Declaration SHALL be considered valid only if it contains both or neither of the dac_origin_vendor_id and dac_origin_product_id fields.

%If the Certification Declaration has both the dac_origin_vendor_id and the dac_origin_produc! t_id fields, the following validation SHALL be done:

& The VendorID value from the subject DN in the DAC SHALL match the dac_origin_ven! dor_id field in the Certification Declaration .

& The VendorID value from the subject DN in the PAI SHALL match the dac_origin_ven! dor_id field in the Certification Declaration .

& The ProductID value from the subject DN in the DAC SHALL match the dac_origin_pro! duct_id field in the Certification Declaration .

& The ProductID value from the subject DN in the PAI, if such a ProductID value appears, SHALL match the dac_origin_product_id field in the Certification Declaration .

%If the Certification Declaration has neither the dac_origin_vendor_id nor the dac_origin_pro! duct_id fields, the following validation SHALL be done:

& The VendorID value from the subject DN in the DAC SHALL match the vendor_id field in the Certification Declaration .

& The VendorID value from the subject DN in the PAI SHALL match the vendor_id field in the Certification Declaration .

& The ProductID value from the subject DN in the DAC SHALL be present in the produc! t_id_array field in the Certification Declaration .

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

- & The ProductID value from the subject DN in the PAI, if such a Product ID is present, SHALL match one of the values present in the product_id_array field in the Certification Declaration .

%If the Certification Declaration contains the authorized_paa_list field, the following valida! tion SHALL be done:

- & The Subject Key Identifier (SKI) extension value of the PAA certificate, which is the root of trust of the DAC, SHALL be present as one of the values in the authorized_paa_list field.

%The certificate_id field SHOULD match the CDCertificateID field found in the entry of the DeviceSoftwareCompliance schema in the Distributed Compliance Ledger where the entryÕs VendorID, Product ID and SoftwareVersion field match the respective VendorID , ProductID and SoftwareVersion attributes values found in the Basic Information Cluster .

%The firmware_information field in the Attestation Information , if present, SHALL match the content of an entry in the Distributed Compliance Ledger for the specific device as explained in Section 6.3.2, ÒFirmware InformationÓ . If the Commissioner does not support Firmware Information validation, it MAY skip checking this match.

The order of execution of the above validation steps MAY be optimized by Commissioners. For example, if some validation steps are deemed by a Commissioner to make the remainder of the steps unnecessary because they have no chance of succeeding, then the validation steps could be ordered such that superfluous steps or rounds trips are omitted.

6.3. Certification Declaration

A Certification Declaration (CD) is a cryptographic document that allows a Matter device to assert its protocol compliance. It is encoded in a CMSformat described in RFCÊ5652Upon successful com! pletion of certification by a device type, Connectivity Standards Alliance creates the CD for that device type so that it can be included in the device firmware by the manufacturer.

6.3.1. Certification Declaration (CD) Format

The Certification Declaration is a CMS [https://tools.ietf.org/html/rfc5652]-encoded single-signature enve! lope whose messageis a TLV-encoded certification-elements structure with an anonymous tag:

Certification Elements TLV structure

```
certification-elements => STRUCTURE [ tag-order ]
{
Ê   format_version [0]      : UNSIGNED INTEGER [ range 16-bits ]
Ê   vendor_id [1]           : UNSIGNED INTEGER [ range 16-bits ]
Ê   product_id_array [2]    : ARRAY [ length 1..100 ] OF UNSIGNED INTEGER [ range 16-
bits ]
Ê   device_type_id [3]      : UNSIGNED INTEGER [ range 32-bits ]
Ê   certificate_id [4]      : STRING [ length 19 ]
Ê   security_level [5]      : UNSIGNED INTEGER [ range 8-bits ]
Ê   security_information [6] : UNSIGNED INTEGER [ range 16-bits ]
Ê   version_number [7]      : UNSIGNED INTEGER [ range 16-bits ]
```

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

```
Ê  certification_type [8]   : UNSIGNED INTEGER [ range 8-bits]
Ê  dac_origin_vendor_id [9, optional]   : UNSIGNED INTEGER [ range 16-bits ]
Ê  dac_origin_product_id [10, optional] : UNSIGNED INTEGER [ range 16-bits ]
Ê  authorized_paa_list [11, optional]   : ARRAY [ length 1..10 ] OF OCTET STRING [
length 20 ]
}
```

The Certification Elements TLV is encoded with data to form a cd_content message to be signed.

```
cd_content =
{
Ê  format_version (0)        = 1,
Ê  vendor_id (1)             = <vendor_id>,
Ê  product_id_array (2)      = <array of product_id values>,
Ê  device_type_id (3)        = <primary device type identifier>,
Ê  certificate_id (4)        = <globally unique certificate ID issued by CSA>,
Ê  security_level (5)        = 0,
Ê  security_information (6)  = 0,
Ê  version_number (7)        = <version_number>,
Ê  certification_type (8)    = <certification_type>,
Ê  dac_origin_vendor_id (9)  = <Vendor ID associated with the DAC, optional>,
Ê  dac_origin_product_id (10) = <Product ID associated with the DAC, optional>,
Ê  authorized_paa_list (11)  = <array of PAA SKIs, optional>
}
```

The format_version field SHALL contain the value 1.

The vendor_id field SHALL contain the Vendor ID associated with the Certification Declaration.

The product_id_array field SHALL contain an array of a number of Product IDs which are covered by the same certification (e.g. certification by similarity). All other fields of a Certification Declara! tion apply to all products in this array.

The device_type_id field SHALL contain the device type identifier for the primary function of the device. For example, if device_type_id is 10 (0x000a), it would indicate that the device has a primary function of a Door Lock device type. See also the _T subtype in Section 4.3.1.3, ÒCommissioning Sub! typesÓ.

The device_type_id field in a given Certification Declaration SHOULD match the device_type_id value in the DCL entries associated with the VendorID and ProductID combinations present in that Certification Declaration.

The certificate_id field SHALL contain a globally unique serial number allocated by the CSA for this Certification Declaration.

The security_level and security_information fields are reserved for future use and SHALL be ignored at read time, and set to zero at issuance time.

The version_number field SHALL contain a version number assigned by the CSA that matches the

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Vendor ID and Product ID used in a DeviceSoftwareVersionModel entry in the Distributed Compli! ance Ledger matching the certification record associated with the product presenting this CD. The value of the version_number is not meant to be interpreted by commissioners and SHALL be recorded as assigned.

The certification_type field SHALL contain the type of certification for this CD, interpreted accord! ing to the following table:

certification_type	meaning
0	used for development and test purposes
1	provisional - used for a device when going into certification testing, or to allow production and distribution to occur in parallel with certifica! tion (with potential software fixes yielding a higher SoftwareVersion which gets certification)
2	official - allocated after passing certification
other values	reserved

For details about the usage of the certification_type field in the Device Attestation Procedure, see failure of Device Attestation Procedure .

The dac_origin_vendor_id field, if present, SHALL contain the Vendor ID value expected to be found in the Device Attestation CertificateÕs subject DN.

The dac_origin_product_id field, if present, SHALL contain the Product ID value expected to be found in the Device Attestation CertificateÕs subject DN.

The dac_origin_vendor_id and dac_origin_product_id SHALL only be present together.

The use of the dac_origin_vendor_id and dac_origin_product_id fields allows for a target of the device attestation procedure to have a manufacturing provenance which differs from the entity that obtains the ultimate certification. If present, they tie a given Certification Declaration to an original manufacturerÕs device attestation chain of trust, so that DACs MAY be issued at manufac! turing time without a priori knowledge of the ultimate vendor.

The optional authorized_paa_list field, if present, SHALL contain a list of one or more Product Attestation Authority (PAA) which is/are authorized (by the device manufacturer) to sign the Prod! uct Attestation Intermediate (PAI) Certificate which signs the Device Attestation Certificate for a product carrying this Certification Declaration. Each such PAA is identified by the Subject Key Iden! tifier (SKI) extension value of its certificate.

Any context-specific tags not listed in the above schema for Certification Elements SHALL be reserved for future use, and SHALL be silently ignored if seen by a Commissioner which cannot understand them.

See Section 6.2.3, ÒDevice Attestation ProcedureÓ for more details about usage of the Certification Declaration fields.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Certification Declaration CMS ASN.1 Encoding Format

```
CertificationDeclaration ::= SEQUENCE {
Ê   version             INTEGER ( v3(3) ),
Ê   digestAlgorithm     OBJECT IDENTIFIER sha256 (2.16.840.1.101.3.4.2.1),
Ê   encapContentInfo    EncapsulatedContentInfo,
Ê   signerInfo          SignerInfo }

EncapsulatedContentInfo ::= SEQUENCE {
Ê   eContentType        OBJECT IDENTIFIER pkcs7-data (1.2.840.113549.1.7.1),
Ê   eContent            OCTET STRING cd_content }

SignerInfo ::= SEQUENCE {
Ê   version             INTEGER ( v3(3) ),
Ê   subjectKeyIdentifier  OCTET STRING,
Ê   digestAlgorithm     OBJECT IDENTIFIER sha256 (2.16.840.1.101.3.4.2.1),
Ê   signatureAlgorithm  OBJECT IDENTIFIER ecdsa-with-SHA256 (1.2.840.10045.4.3.2),
Ê   signature           OCTET STRING }
```

The Certification Declaration encoding rules:

1. The format SHALL only support CMS version v3.
2. The digestAlgorithm SHALL use the sha256algorithm.
3. The signatureAlgorithm SHALL use the ecdsa-with-SHA256(ECDSA with SHA256) and secp256r1 curve, as defined in Section 2.4.2 of SECÊ2
4. The eContentTypeSHALL use the pkcs7-data type.
5. The subjectKeyIdentifier SHALL contain the subject key identifier (SKI) of a well-known Con! nectivity Standards Alliance certificate, that was used to generate the signature . The format of the key identifiers supported is available as part of the Certification Policy.

Note that Certification Declarations SHALL NOT be generated by any Node, but rather, they SHALL be stored and transmitted to a Commissioner by a Commissionee during the conveyance of the Attestation Information in response to an Attestation Request command .

See Section F.1, ÒCertification Declaration CMS test vectorÓ for a complete example of generating a Certification Declaration.

6.3.2. Firmware Information

Firmware Information is an optional component of the Device Attestation Information (see Section 6.2.3, ÒDevice Attestation ProcedureÓ).

Firmware Information MAY contain one or more Firmware Digests that correspond to the compo! nents in the firmware that have been measured and recorded during the boot process (e.g., boot! loader, kernel, root filesystem, etc), and MAY contain other metadata. A Firmware Digest SHALL either represent a hash of the corresponding firmware layer or a hash of the signed manifest that was used to validate the corresponding firmware layer during secure boot. An implementation MAY choose to hash the measurements of all components into a single hash and include only that

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

hash in Firmware Information.

A device MAY report Firmware Information containing its firmware digests only if it implements a secure subsystem that protects the device attestation private key and is able to securely collect and report firmware digests as shown in Figure 35, ÒIllustration of the measured boot processÓ . This process is known as "measured boot".

Ideally, the measured boot process SHOULD be rooted in silicon such as a boot ROM, similar to the secure boot process found in many systems-on-chip (SoCs). Since many SoCs and microcontrollers are unable to perform measured boot in hardware, the process SHOULD start at the earliest firmware component possible (for example, at the bootloader shown in the figure below). In this case, this firmware component is not measured and in fact, it is the root for measurement. There! fore, it SHALL be resistant to attacks compromising subsequent firmware components (e.g., the ROM must verify its authenticity (secure boot) or it may be placed in a locked partition at the fac! tory that cannot be updated by software in the field).

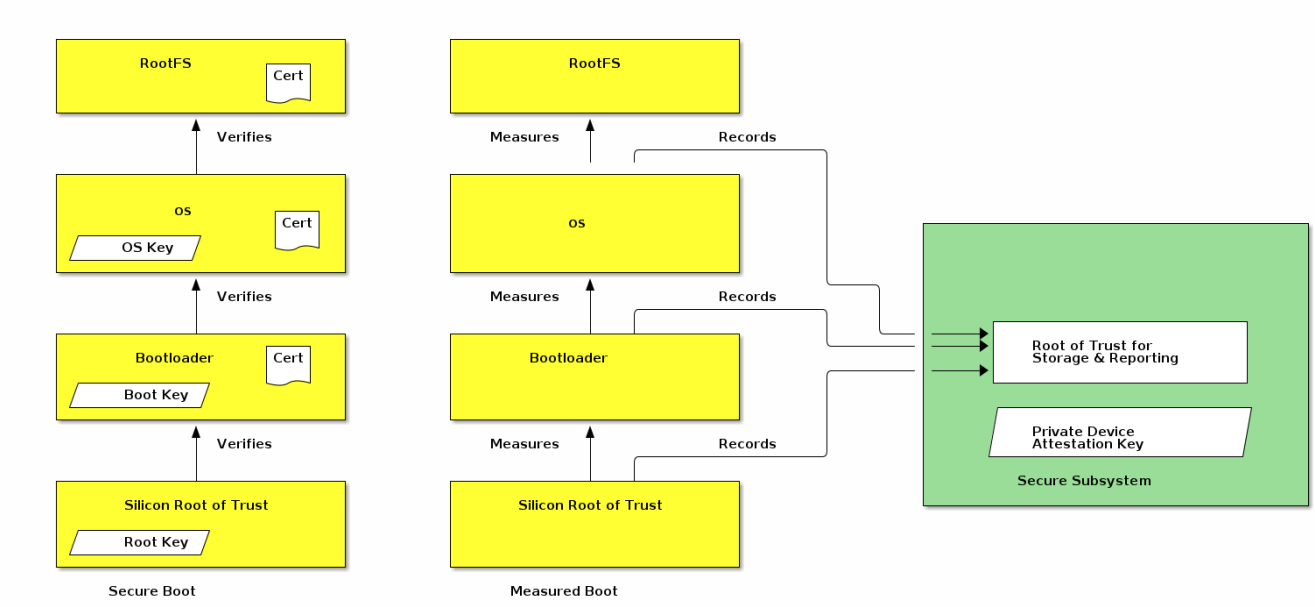


Figure 35. Illustration of the measured boot process

The device secure subsystem SHALL use the device attestation private key to sign attestation-ele! ments and NOCSR-elements. The device secure subsystem SHALL fill the attestation-elements fields using information compiled into its image or generated during the measured boot process. The device secure subsystem SHALL validate all signing requests so that if the device software, but not its secure subsystem, gets compromised it cannot act as a signing oracle to sign Attestation Informa! tion Responses with fake Firmware Digests.

The firmware_information field in attestation-elements SHALL NOT be generated by devices that do not implement a separate secure subsystem, in software or hardware, which maintains and con! trols the use of the device attestation private key.

For devices that support secure boot, it is straightforward to add support for measured boot. Specif! ically, the hashes of the different firmware components that are already generated and verified sequentially during secure boot SHALL be collected and stored for reporting. Devices that do not support secure boot MAY implement measured boot by generating the hashes in software during the boot process implementing the root for measurement in the earliest firmware component.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

If a device that chooses to send Firmware Digests and which supports an industry-standard measured boot architecture and which can generate signed firmware attestation reports, the secure subsystem in the device MAY validate the firmware attestation reports locally and SHALL report the raw firmware digests in attestation-elements so that the firmware_information field in attestation-elements has the same values in all devices of the same model that run the specific Software Image.

Firmware Digests SHALL NOT be reported by devices that implement a single firmware component in the boot chain, because there is nothing to measure and report subsequently, unless they have support for measured boot built in the deviceÕs boot ROM.

Commissioners MAY use the reported firmware information to confirm that the firmware version is authorized to run on the device, that it has not been revoked, or that it does not contain known vulnerabilities. Commissioners and Administrators that choose to verify this information SHOULD refer to canonical databases, such as the Distributed Compliance Ledger (see Section 11.22, ÒDistributed Compliance LedgerÓ) to validate that the reported firmware information matches what is expected for an authorized Software Image associated with a given Certification Declaration. The firmware information, when validated, SHALL be validated as an opaque well-known octet string. Internal semantic validation MAY be applied for error-reporting, but the exact format is out of the scope of this specification.

In cases where a Commissioner or Administrator detects such an invalid or problematic firmware version, Commissioners and Administrators MAY, after consultation with the user, refuse to commission the device, provide it with operational credentials, or otherwise operate it, until the firmware has been updated, to avoid putting the user at risk from compromised software.

6.3.3. Firmware information validation examples

Below is an illustrative example of the Commissioner actions to validate the firmware information.

1. Retrieve the firmware_information field from attestation-elements
2. Retrieve all Distributed Compliance Ledger DeviceSoftwareVersionModel entries for the CommissioneeÕs Vendor ID and Product ID.
3. Verify that there is a valid, non-revoked, entry where the FirmwareInformation field exactly matches the firmware_information field in attestation-elements .
4. If verification fails, report error to the user
5. If verification succeeds, proceed with device commissioning

Below is an example of the corresponding Device actions. For illustrative purposes, it is assumed that the device implements a secure subsystem that maintains the private device attestation key and signs attestation-elements using this key but it does not have direct hardware support for measured boot. This is expected to be the common case for many devices covered by this version of the specification. Consequently, the measurement process can only start from the bootloader shown in the figure above.

1. The device bootloader produces a measurement of the OS kernel using a supported hash algorithm from RFC 5912 and delivers it to the secure subsystem.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

2. The secure subsystem receives the measurement and stores in a location inaccessible to the OS.
3. The OS kernel produces a hash of the root filesystem and delivers the measurement to the secure subsystem.
4. When the secure subsystem is asked to sign an attestation-elements structure using its private device attestation key, it generates two FirmwareDigests or one combined FirmwareDigest from these measurements, fills the firmware_information field in attestation-elements using these measurements, fills the CD blob compiled into the secure environment and signs the attestation-elements structure.

The Device Vendor is responsible to provide the FirmwareInformation field when a new Software Image entry is reported in the corresponding Distributed Compliance Ledger entry.

Below is an exemplary ASN.1 schema for an encoding scheme that could be used to encode firmware information.

Firmware Information encoding example

```
HashAlgorithm ::= SEQUENCE {
Ê id OBJECT IDENTIFIER,
Ê params ANY OPTIONAL
}

FirmwareDigest ::= SEQUENCE {
Ê digestAlgorithm  HashAlgorithm,
Ê digestHash       OCTET STRING
}

FirmwareInformation ::= SEQUENCE {
Ê firmwareDigests  SEQUENCE OF FirmwareDigest
}

-- Example HashAlgorithm id
id-sha256  OBJECT IDENTIFIER  ::= {
Ê joint-iso-itu-t(2) country(16) us(840) organization(1) gov(101)
Ê csor(3) nistAlgorithms(4) hashalgs(2) 1
}

-- Below is an example value for the above exemplary FirmwareInformation

firmwareInformation FirmwareInformation ::= {
Ê -- The firmwareDigests contain two values, for two separate components.
Ê firmwareDigests {
Ê   {
Ê     digestAlgorithm {
Ê       id id-sha256
Ê     },
Ê     digestHash '00112233445566778899AABBCCDDEEFF00112233445566778899AABBCCDDEEFF'H
Ê   },
Ê   {
Ê     digestAlgorithm {
```


Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

```
Ê      id id-sha256
Ê    },
Ê    digestHash '101112131415161718191A1B1C1D1E1F101112131415161718191A1B1C1D1E1F'H
Ê  }
Ê }
}
```

The above example would yield the following DER-encoded octet string:

```
30663031 300d0609 60864801 65030402 01050004 20001122 33445566 778899AA
BBCCDDEE FF001122 33445566 778899AA BBCCDDEE FF303130 0D060960 86480165
03040201 05000420 10111213 14151617 18191A1B 1C1D1E1F 10111213 14151617
18191A1B 1C1D1E1F
```

6.4. Node Operational Credentials Specification

6.4.1. Introduction

The Node Operational credentials are a collection of credentials to enable a Node to identify itself within a Fabric. The Node Operational credentials are distinct from the Device Attestation creden! tials. The Node Operational credentials are installed during Commissioning .

The Node Operational credentials include the following items:

- ¥ Node Operational Key Pair
- ¥ Node Operational Certificate (NOC)
- ¥ Intermediate Certificate Authority (ICA) Certificate (optional)
- ¥ Trusted Root Certificate Authority (CA) Certificate(s)

Each Node in a Fabric is identified with a Node Operational Identifier . In order to securely identify the Node, the Node Operational Identifier is bound to the Node Operational Public Key as both are contained within the signed NOC. The Node Operational Identifier is a constituent part of the sub! ject field of the NOC, according to the rules described in Matter DN Encoding Rules . A connecting Node can attest to the validity of the Node Operational Public Key and the Node Operational Identi! fier in a received NOC because the NOC is signed by a CA that the connecting Node trusts. Used with Certificate Authenticated Session Establishment (CASE) , these data provide the basis for secure communications on the Fabric.

6.4.2. Node Operational Credentials Management

Commands from the Node Operational Credentials Cluster are used to install and update Node Operational credentials.

A Node receives its initial set of Node Operational credentials through the AddNOC command when it is commissioned onto a Fabric by a Commissioner.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Once installed, Node Operational credentials MAY be updated by an Administrator with the appro! priate privileges using the UpdateNOC command.

Once installed, Node Operational credentials MAY be removed by an Administrator with the appro! priate privileges using the RemoveFabric command. The removal uses RemoveFabric, since the Fabric association for the given Node Operational credentials may underpin a variety of bindings and other fabric-scoped configuration, which would remain in an inconsistent state if the Node Opera! tional credentials alone were removed, as opposed to the entire associated Fabric and data.

6.4.3. Node Operational Identifier Composition

The Node Operational Identifier is used for Node discovery and network address resolution within a network segment. The FabricID portion of the Node Operational Identifier serves a scoping pur! pose to identify disjoint operational Fabrics within a given network segment. The NodeIDportion of the Node Operational Identifier is the logical addressing identifier used:

- within Message-layer messages for logical addressing (see Section 4.4, ÒMessage Frame FormatÓ)
- within Data Model bindings to express data subscription relationships between Nodes (see Sys! tem Model)
- within Access Control List Entries to refer to individual Nodes as access control grantees (sub! jects) when CASEsessions are used for communication (see Access Control Cluster)

In addition to the FabricID and NodeID, a Node Operational Identifier MAY include at most three 32-bit CASEAuthenticatedTag(1.3.6.1.4.1.37244.1.6) attributes used to tag the operational identifier to implement access control based on CASE Authenticated Tags .

The Fabric ID is a 64-bit value that identifies the Fabric and is scoped to a particular Root CA. For example, two fabrics with the same Fabric ID are not equivalent unless their Root CA are the same. The Fabric ID MAY be chosen randomly or algorithmically but it SHALL be allocated uniquely within the set of all possible Fabric IDs for which a given Root CA will sign operational certificates. Before allocating the Fabric ID, the Commissioner SHOULD attempt to ensure that an existing Fab! ric is reused and joined, if any is applicable from the perspective of the Commissioner in the cur! rent commissioning context. The method used for determining local Fabric ID existence is vendor-specific.

The Node ID is a 64-bit value that identifies a Node within a Fabric. The Node ID MAY be chosen randomly or algorithmically but it SHALL be allocated uniquely within the Fabric before it is given to the Node or otherwise used. The Node ID SHALL be chosen, by a Commissioner, at the time of Node commissioning.

The uniqueness constraint for Fabric ID is only required to be ensured within the scope of the Root CA serving the Commissioner.

When a Fabric is removed, through the RemoveFabric command or through a factory reset, the Node Operational Identifier, and the FabricID and NodeIDthat comprise it, SHALL be permanently removed from the NodeÕs memory.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

6.4.4. Node Operational Key Pair

A Node Operational Key Pair, comprised of a Node Operational Public Key and a Node Operational Private Key, is created using the Crypto_GenerateKeypair function. A new Node Operational Key Pair is generated for each Commissioning Session in accordance with security requirements .

6.4.5. Node Operational Credentials Certificates

All certificates in the Node Operational credentials are X.509v3 certificates compliant with RFC 5280, encoded in such a way that they respect the constraints in the Operational_Certificate section. They may be encoded as X.509v3 certificates or Matter Operational Certificates ("Matter Certificates" thereafter). The signature field of a certificate SHALL be calculated using the X.509v3 encoding of the certificate.

6.4.5.1. Node Operational Certificate (NOC)

The NOC SHALL be issued by either a Root CA trusted within the Fabric or by an Intermediate Certificate Authority (ICA) whose ICA certificate is directly issued by such a Root CA. The NOC is bound to the Node Operational Key Pair through the Node Operational Credential Signing Request (NOCSR).

The validity period specifies the time period for which a NOC is valid. For constrained or sleepy devices that lack accurate time, enforcement of an NOC's validity period MAY be omitted.

6.4.5.2. Intermediate CA (ICA) Certificate

In the case where an intermediate CA (ICA) issues the NOC, the ICA certificate is used to attest to the validity of the NOC. The Root CA certificate associated with the issuer of the ICA certificate is used in turn to attest to the validity of the ICA certificate.

6.4.5.3. Trusted Root CA Certificates

Each Node has one or more trusted Root CA certificates in its Node Operational credentials that it uses to verify ICA certificates and Node Operational Certificates presented by other Nodes, treating them as trust anchors as described in RFC 5280. A Root CA certificate is self-signed. They are not verified but rather trusted because they were provisioned by a trusted Commissioner.

In the case where a Root CA issues the NOC, the Root CA certificate is used to attest to the validity of the NOC.

The trusted Root CA certificates that a Device trusts when the Device is verifying operational certificates are those stored in the TrustedRootCertificates attribute of that Device's Node Operational Credentials cluster .

A device MAY have Root CA certificates that it trusts for purposes other than for operational credential verification. These certificates SHALL NOT appear in any Node's TrustedRootCertificates attribute of the Node Operational Credentials cluster . The certificates configured in that cluster SHALL only be added during the commissioning process by the Commissioner, or during root rotation operations by an Administrator already trusted by the Node. Nodes SHALL NOT modify the TrustedRootCertificates attribute outside of the processing of Node Operational Credentials cluster

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

commands.

The figures below show the Node Operational Certificate hierarchies, with and without optional ICAC.

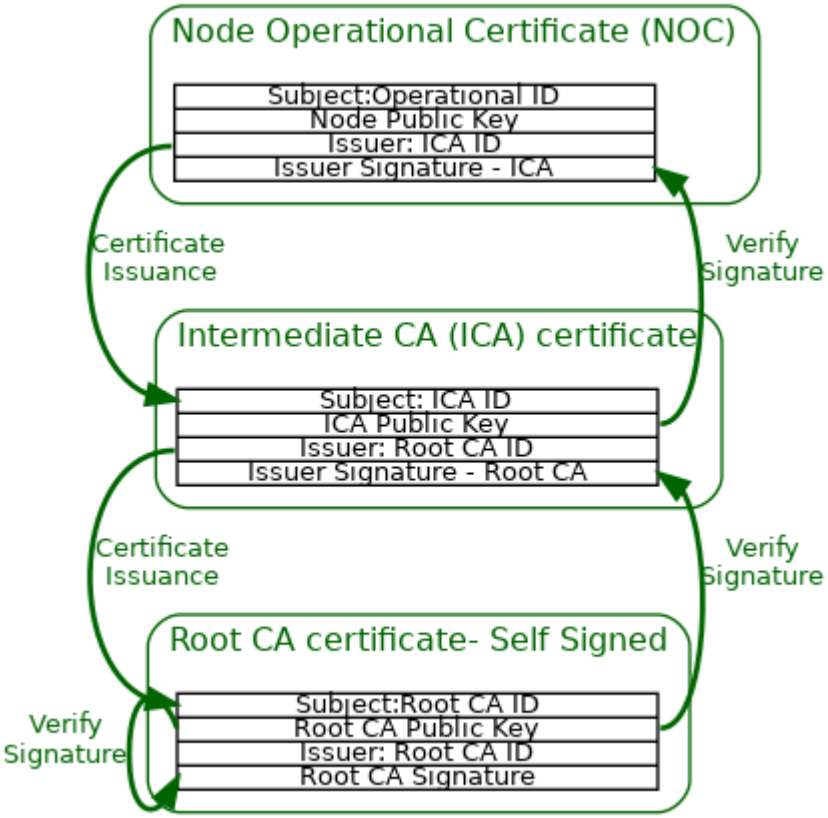


Figure 36. Node Operational Certificate PKI hierarchy with optional ICAC

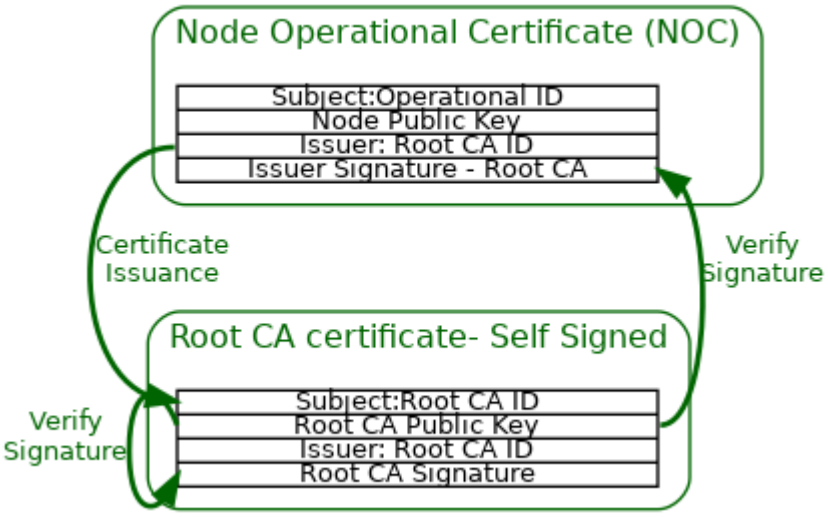


Figure 37. Node Operational Certificate PKI hierarchy without optional ICAC

6.4.6. Node Operational Credentials Procedure

The following procedure is used by a Node to obtain an Operational Credential. This procedure is part of Commissioning .

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

6.4.6.1. Node Operational Certificate Signing Request (NOCSR) Procedure

After the Commissioner validates Device Attestation Information, the following procedure is used to generate a Node Operational Key Pair and obtain the NOCSR.

1. The Commissioner SHALL generate a random 32 byte nonce named CSRNonce using Crypto_DRBG().
2. The Commissioner SHALL send the CSRNonce to the Node and request NOCSR Information using the CSRRequest Command.
   a. The Node SHALL create a new candidate Node Operational Key Pair, using Crypto_GenerateKeyPair() , valid for the duration of the Fail-Safe Context currently in progress.
   b. The Node SHOULD verify that the newly generated candidate Node Operational Key Pair does not match any other existing Node Operational Key Pair on the device. If such a key collision was to be found, it would indicate a key pair that was not properly randomly generated. The procedure SHALL fail if such a collision is detected. See Section 11.17.7.5, ÒCSRRequest CommandÓ for the error generated in that situation.
   c. The candidate Node Operational Key Pair SHALL only be committed to persistent storage upon successful execution of the next AddNOC Command executed with a Node Operational Certificate whose public key matches the candidate key.
   d. The Node SHALL create a Certificate Signing Request (CSR) by following the format and procedure in PKCS #10, which includes a signature using the Node Operational Private Key (see RFC 2986 section 4.2).
   e. The CSRÕs subject MAY be any value and the device SHOULD NOT expect the final operational certificate to contain any of the CSRÕs subject DN attributes.
3. The Node SHALL generate and return the NOCSR Information (see Section 11.17.5.7, ÒNOCSR InformationÓ for encoding) to the Commissioner using the CSRResponse Command. The NOCSR Information includes a signature using the Device Attestation Private Key.

Node Operational CSR Information Validation

1. The Commissioner SHALL validate the Device Attestation Signature (attestation_signature) field from CSRResponse Command:

```
Success = Crypto_Verify(
Ê  publicKey = Public key from DAC,
Ê  message = NOCSR Information TBS (nocsr_tbs),
Ê  signature = Device Attestation Signature ( attestation_signature )
)
```

   where the fields are encoded as described in Section 11.17.5.7, ÒNOCSR InformationÓ.

   %The AttestationChallenge SHALL be obtained from a CASE session, resumed CASE session, or PASE session depending on the method used to establish the secure session within which device attestation is conducted.

   %The CSR Nonce in NOCSR Information SHALL match the CommissionerÕs CSR Nonce.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

2. The inner signature in the PKCS#10 csr sub-field of the CSRResponse Command's NOCSRElements field SHALL be verified, per the definition of CSR signatures in PKCS #10.

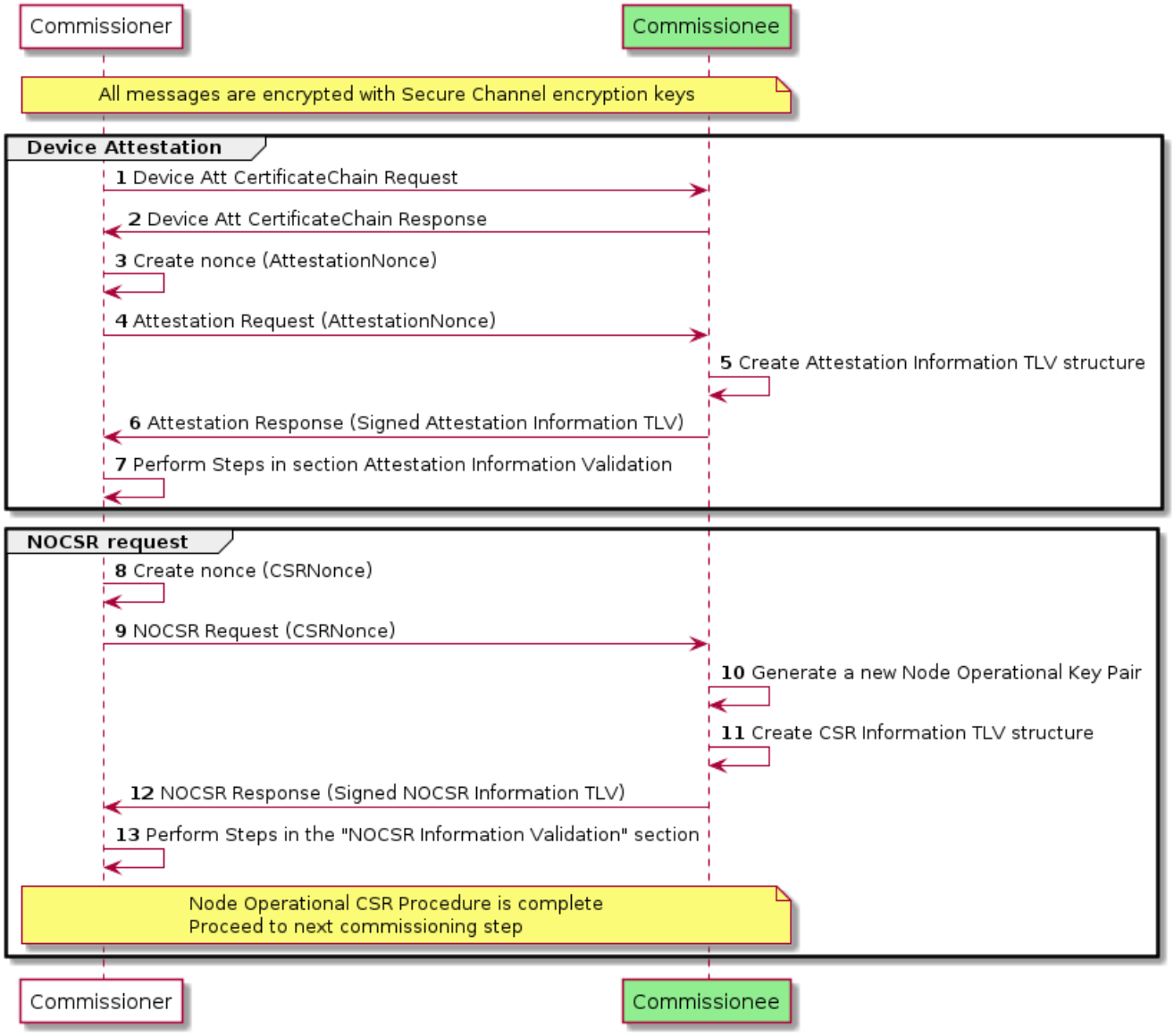


Figure 38. Node Operational Credentials flow

6.4.7. Node Operational Certificate Signing Request (NOCSR)

A Node creates a NOCSR in response to the Commissioner, so that the Commissioner can request a NOC on the NodeÕs behalf from its trusted Certificate Authority. The CSR itself SHALL follow the encoding and rules from PKCS #10 with the minimum attributes shown in the example below.

Note that the subject field MAY be any value.

NOCSR

```
Certificate Request:
Ê   Data:
Ê      Version: 1 (0x0)
Ê      Subject: .....
Ê      Subject Public Key Info:
Ê         Public Key Algorithm: id-ecPublicKey
```

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

```
Ê            Public-Key: (256 bit)
Ê            pub:
Ê               04:12:3b:90:f5:.....
Ê            ASN1 OID: prime256v1
Ê            NIST CURVE: P-256
Ê      Attributes:
Ê      Requested Extensions:
Ê  Signature Algorithm: ecdsa-with-SHA256
Ê       30:46:02:21:00:95:ff:.....
```

6.4.8. Node Operational Certificate Renewal

A NOC can be renewed by an Administrator (a Node that has Administer privileges on the Node to be updated). The Administrator triggers the process by sending an CSRRequest Command.

6.4.9. Node Operational Certificate Revocation

A NodeÕs access to other Nodes can be revoked by removing the associated Node ID from Access Control Entry subjects where it appears. This action is taken by an Administrator which has the privilege to update the Access Control Cluster for its Nodes.

6.4.10. Security Considerations

A NOC is a NodeÕs credential to operate on a Fabric. It SHALL be protected against the following threats:

1. The Node Operational Private Key SHALL be protected from unauthorized access.
2. The Node Operational Private Key SHOULD never leave the device.
3. The NOC SHALL NOT contain information that may violate the userÕs privacy.
4. The NOC SHALL be wiped if the Node is factory reset.

6.5. Operational Certificate Encoding

6.5.1. Introduction

This section details the Matter certificate data structure (hereafter "Matter certificate"), a specific encoding that is sometimes used as a compact alternative to the standard X.509 certificate format [RFCÊ5280] for bandwidth-efficient transmission. A Node Operational Certificate (NOC) , Intermediate CA certificate and Root CA certificate MAY all be encoded as a Matter certificate.

To compress the structure more efficiently than an X.509 certificate, a Matter certificate SHALL be encoded with the Matter TLV structured data interchange language [Appendix A, Tag-length-value (TLV) Encoding Format] instead of the ASN.1 Distinguished Encoding Rules (DER) [X.690].

This section provides a technical specification of the structure of data comprising a Matter certificate with accompanying requirements for their semantic validation, and their conversion to and from X.509 certificates. In some cases, as noted, the limitations on the semantic interpretation of

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

parts of a Matter certificate follow from limitations applied by [RFCÊ5280].

A certificate comprises a record of the following conceptual fields:

Certificate Text
Ê Version Number
Ê Serial Number
Ê Signature Algorithm ID
Ê Issuer Name
Ê Validity period
Ê Not Before
Ê Not After
Ê Subject name
Ê Subject Public Key Info
Ê Public Key Algorithm
Ê Subject Public Key
Ê Issuer Unique Identifier
Ê Subject Unique Identifier
Ê Extensions
Certificate Signature Algorithm
Certificate Signature

6.5.1.1. ASN.1 Object Identifiers (OID)

Several important components of X.509 certificates follow the pattern commonly used in ASN.1 data models where some types are constructed with an ASN.1 object identifier (OID) to identify each variant. For example, the cryptographic algorithm used in the digital signature is identified by its OID.

Matter certificates do not use ASN.1 OIDs. Instead, each valid ASN.1 OID SHALL be mapped to a Matter TLV tag within its reference category. Each reference category defines the context of the Matter tag, and tag values are assigned to the reference categories according to the type of fields where they can appear in X.509 certificates.

6.5.2. Matter certificate

A Matter certificate encodes a subset of the object identifiers (OIDs) specified in X.509. Only some attribute types for relative distinguished names are valid, only certain cryptographic algorithms (corresponding to the algorithms as defined in Chapter 3, Cryptographic Primitives) are used, and only a limited set of extensions are used. Therefore, every Matter certificate can be represented as a corresponding X.509 certificate. However, the converse is not true; not every X.509 certificate can be represented as a Matter certificate.

The signature included in a Matter certificate is the signatureValue of the corresponding X.509 cer! tificate, not a signature of the preceding Matter TLV data in the Matter certificate structure. Accord! ingly, validating the signature in a Matter certificate entails its logical conversion to the correspond! ing X.509 certificate to recover the original tbsCertificate of the basic syntax signed by the Certifi! cate Authority (CA).

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

```
matter-certificate [anonymous] => STRUCTURE [tag-order]
{
Ê  serial-num [1]     : OCTET STRING [ length 0..20 ],
Ê  sig-algo [2]       : signature-algorithm,
Ê  issuer [3]         : LIST [ length 1.. ] OF dn-attribute,
Ê  not-before [4]     : UNSIGNED INTEGER [ range 32-bits ],
Ê  not-after [5]      : UNSIGNED INTEGER [ range 32-bits ],
Ê  subject [6]        : LIST [ length 1.. ] OF dn-attribute,
Ê  pub-key-algo [7]   : public-key-algorithm,
Ê  ec-curve-id [8]    : elliptic-curve-id,
Ê  ec-pub-key [9]     : OCTET STRING,
Ê  extensions [10]    : LIST [ length 1.. ] OF extension,
Ê  signature [11]     : ec-signature,
}
```

6.5.3. Version Number

Matter certificates SHALL only support version X.509 v3. This field is not encoded in the Matter cer! tificate structure.

6.5.4. Serial Number

The context-specific tag serial-num [1] SHALL be used to identify the serial number field in the Matter certificate structure.

A Matter certificate follows the same limitation on admissible serial numbers as in [RFCÊ5280], i.e., that implementations SHALL admit serial numbers up to 20 octets in length, and certificate authori! ties SHALL NOT use serial numbers longer than 20 octets in length.

6.5.5. Signature Algorithm

Like an X.509 certificate, a Matter certificate SHALL include a digital signature in its signature com! ponent. The signature algorithm component of a Matter certificate specifies the cryptographic algo! rithm used for composing and validating the signature embedded in the signature component of the certificate. The signature algorithm SHALL match the algorithm in Section 3.5.3, ÒSignature and verificationÓ .

The context-specific tag sig-algo [2] SHALL be used to identify the signature algorithm field in the Matter certificate structure.

```
signature-algorithm => UNSIGNED INTEGER [ range 8-bits ]
{
Ê  ecdsa-with-sha256   = 1,
}
```

The following values SHALL be defined for signature-algorithm :

Table 55. Signature Algorithm Object Identifiers

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Value	ASN.1 OID
1	iso(1) member-body(2) us(840) ansi-x962(10045) signatures(4) ecdsa-with-SHA2(3) ecdsa-with-SHA256(2)

6.5.6. Issuer and Subject

The context-specific tags issuer [3] and subject [6] SHALL be used to identify the issuer and the subject DN fields in the Matter certificate structure. The entries in the lists SHALL be Distinguished Names (DNs), which are described in Section 6.5.6.1, ÒX.501 Distinguished NamesÓ.

6.5.6.1. X.501 Distinguished Names

The Issuer Name and Subject Name components of an X.509 certificate contain DNs as defined in [RFCÊ5280]. The ASN.1 format of a DN is a sequence of Relative Distinguished Names (RDNs). Two distinguished names DN1 and DN2 match if they have the same number of RDNs, for each RDN in DN1 there is a matching RDN in DN2, and the matching RDNs appear in the same order in both DNs.

The RDN in an X.509 certificate may be encoded as a set of one or more DN attributes, although in practice it is usually a single DN attribute. The RDN in a Matter certificate SHALL be always a single DN attribute. Two relative distinguished names RDN1 and RDN2 match if the attribute in RDN1 matches the attribute in RDN2.

```
dn-attribute => CHOICE OF
{
Ê  // Standard and Matter-specific DN attributes.
Ê  // Of these, all are encoded as UTF8String except domain-component,
Ê  // which is encoded as IA5String in X.509 form.
Ê  common-name [1]               : STRING,
Ê  surname [2]                   : STRING,
Ê  serial-num [3]                : STRING,
Ê  country-name [4]              : STRING,
Ê  locality-name [5]             : STRING,
Ê  state-or-province-name [6]     : STRING,
Ê  org-name [7]                  : STRING,
Ê  org-unit-name [8]             : STRING,
Ê  title [9]                     : STRING,
Ê  name [10]                     : STRING,
Ê  given-name [11]               : STRING,
Ê  initials [12]                 : STRING,
Ê  gen-qualifier [13]            : STRING,
Ê  dn-qualifier [14]             : STRING,
Ê  pseudonym [15]                : STRING,
Ê  domain-component [16]         : STRING,
Ê  matter-node-id [17]           : UNSIGNED INTEGER,
Ê  matter-firmware-signing-id [18] : UNSIGNED INTEGER,
Ê  matter-icac-id [19]           : UNSIGNED INTEGER,
Ê  matter-rcac-id [20]           : UNSIGNED INTEGER,
Ê  matter-fabric-id [21]         : UNSIGNED INTEGER,
Ê  matter-noc-cat [22]           : UNSIGNED INTEGER,
```


Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

```
Ê  // Standard DN attributes when encoded as PrintableString in X.509 form
Ê  // NOTE: The tags for these SHALL be the base tags + 0x80.
Ê  common-name-ps [129]           : STRING,
Ê  surname-ps [130]               : STRING,
Ê  serial-num-ps [131]            : STRING,
Ê  country-name-ps [132]          : STRING,
Ê  locality-name-ps [133]         : STRING,
Ê  state-or-province-name-ps [134] : STRING,
Ê  org-name-ps [135]              : STRING,
Ê  org-unit-name-ps [136]         : STRING,
Ê  title-ps [137]                 : STRING,
Ê  name-ps [138]                  : STRING,
Ê  given-name-ps [139]            : STRING,
Ê  initials-ps [140]              : STRING,
Ê  gen-qualifier-ps [141]         : STRING,
Ê  dn-qualifier-ps [142]          : STRING,
Ê  pseudonym-ps [143]             : STRING,
}
```

Table 56, ÒStandard DN Object IdentifiersÓ lists the context-specific tags defined for the standard DN attribute types used in Matter that can be encoded in X.509 certificates as either UTF8String or as PrintableString format. In Matter certificates, the context-specific tag is logically-ORed with 0x80 (and its name given a corresponding -ps suffix) to indicate that the corresponding X.509 encoding of the attribute uses the PrintableString format instead of UTF8String.

Table 56. Standard DN Object Identifiers

Tag base	Matter name base	ASN.1 OID
1	common-name	joint_iso_ccitt(2) ds(5) attributeType(4) commonName(3)
2	surname	joint_iso_ccitt(2) ds(5) attributeType(4) surname(4)
3	serial-num	joint_iso_ccitt(2) ds(5) attributeType(4) serialNumber(5)
4	country-name	joint_iso_ccitt(2) ds(5) attributeType(4) countryName(6)
5	locality-name	joint_iso_ccitt(2) ds(5) attributeType(4) localityName(7)
6	state-or-province-name	joint_iso_ccitt(2) ds(5) attributeType(4) stateOrProvinceName(8)
7	org-name	joint_iso_ccitt(2) ds(5) attributeType(4) organizationName(10)
8	org-unit-name	joint_iso_ccitt(2) ds(5) attributeType(4) organizationalUnit! Name(11)
9	title	joint_iso_ccitt(2) ds(5) attributeType(4) title(12)
10	name	joint_iso_ccitt(2) ds(5) attributeType(4) name(41)
11	given-name	joint_iso_ccitt(2) ds(5) attributeType(4) givenName(42)
12	initials	joint_iso_ccitt(2) ds(5) attributeType(4) initials(43)
13	gen-qualifier	joint_iso_ccitt(2) ds(5) attributeType(4) generationQualifier(44)

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Tag base	Matter name base	ASN.1 OID
14	dn-qualifier	joint_iso_ccitt(2) ds(5) attributeType(4) dnQualifier(46)
15	pseudonym	joint_iso_ccitt(2) ds(5) attributeType(4) pseudonym(65)

Table 57, ÒStandard DN Domain Component Object IdentifierÓ lists the context-specific tag defined for the standard DN attribute type used in Matter that is encoded in X.509 certificates as IA5String .

Table 57. Standard DN Domain Component Object Identifier

Tag	Matter name	ASN.1 OID
16	domain-component	itu_t(0) data(9) pss(2342) ucl(19200300) pilot(100) pilotAttribute! Type(1) domainComponent(25)

In addition to the standard DN attribute types, there are Matter-specific DN attribute types under the 1.3.6.1.4.1.37244 private arc. See Section 6.1.1, ÒEncoding of Matter-specific RDNsÓ for con! straints and examples related to usage of Matter-specific DN attribute types.

6.5.6.2. Matter Certificate Types

The Matter-specific DN attribute types convey information about Matter-specific certificate types as listed in Table 58, ÒMatter Certificate TypesÓ .

Table 58. Matter Certificate Types

Matter name	Description
matter-node-id	Certifies the identity of a Matter Node Operational Certificate (NOC) .
matter-firmware-signing-id	Certifies the identity of a firmware signing certificate.
matter-icac-id	Certifies the identity of a Matter Intermediate CA (ICA) Certificate .
matter-rcac-id	Certifies the identity of a Matter Root CA Certificate .

The value of matter-icac-id and matter-rcac-id DN attribute types MAY be any 64-bit identifier desired by the certificateÕs issuer. Apart from marking what type of certificates are involved, they MAY be used for debugging purposes to determine the specific CA in use, for example if different production tiers or regions are used.

6.5.6.3. Matter DN Encoding Rules

The rules that SHALL be followed for Matter-specific attribute types when encoding the subject DN are:

¥ For a Matter Node Operational Certificate (NOC) :

%The subject DN SHALL encode exactly one matter-node-id attribute.

& The matter-node-id attributeÕs value SHALL be in the Operational Node ID range (0x0000_0000_0000_0001 to 0xFFFF_FFEF_FFFF_FFFF), see Table 4, ÒNode Identifier Allo! cationsÓ.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

- %The subject DN SHALL encode exactly one matter-fabric-id attribute.
  - & The matter-fabric-id attributeÕs value SHALL NOT be 0 (see Section 2.5.1, ÒFabric References and Fabric IdentifierÓ).
- %The subject DN SHALL NOT encode any matter-icac-id attribute.
- %The subject DN SHALL NOT encode any matter-rcac-id attribute.
- %The subject DN MAY encode at most three matter-noc-cat attributes.
  - & Each matter-noc-cat attribute present, if any, SHALL encode a different CASE Authenticated Tag identifier (upper 16 bits of value) than is used by other matter-noc-cat attributes (CATs).

¥ For a Matter ICA Certificate :

- %The subject DN SHALL NOT encode any matter-node-id attribute.
- %The subject DN MAY encode at most one matter-fabric-id attribute.
  - & If present, the matter-fabric-id attributeÕs value SHALL NOT be 0 (see Section 2.5.1, ÒFabric References and Fabric IdentifierÓ).
- %The subject DN MAY encode at most one matter-icac-id attribute.
- %The subject DN SHALL NOT encode any matter-rcac-id attribute.
- %The subject DN SHALL NOT encode any matter-noc-cat attribute.

¥ For a Matter Root CA Certificate :

- %The subject DN SHALL NOT encode any matter-node-id attribute.
- %The subject DN MAY encode at most one matter-fabric-id attribute.
  - & If present, the matter-fabric-id attributeÕs value SHALL NOT be 0 (see Section 2.5.1, ÒFabric References and Fabric IdentifierÓ).
- %The subject DN SHALL NOT encode any matter-icac-id attribute.
- %The subject DN MAY encode at most one matter-rcac-id attribute.
- %The subject DN SHALL NOT encode any matter-noc-cat attribute.

¥ The attributes SHALL appear in the same order in the Matter certificate and in the corresponding X.509 certificates.

¥ When any matter-fabric-id attributes are present in either the Matter Root CA Certificate or the Matter ICA Certificate , the value SHALL match the one present in the Matter Node Operational Certificate (NOC) within the same certificate chain.

¥ The order of the attributes can be issuer-specific and is not enforced by Matter specifications.

¥ All implementations SHALL accept, parse, and handle Matter certificates with up to 5 RDNs in a single DN.

¥ All implementations SHALL reject Matter certificates with more than 5 RDNs in a single DN.

In addition to the above rules, the encoding constraints in Section 6.1.1, ÒEncoding of Matter-specific RDNsÓ SHALL be followed.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

6.5.6.4. Matter DN Examples

The following is an example of subject DN encoding for a Matter Node Operational Certificate (NOC). Typically, it is a list of two RDN attributes:

```
subject = [[
Ê   matter-node-id   = 0x0102030405060708U,
Ê   matter-fabric-id = 0xFAB000000000001DU
]]
```

In addition to the mandatory attributes, it may also encode other supported RDN attributes such as common-name and CASE Authenticated Tags as presented below:

```
subject = [[
Ê   common-name      = "NOC Example",
Ê   matter-node-id   = 0x0102030405060708U,
Ê   matter-fabric-id = 0xFAB000000000001DU,
Ê   matter-noc-cat   = 0xABCD0002U
]]
```

The following subject DN example illustrates that multiple RDN attributes of the same type can be encoded. The specific order of attributes is not enforced. Note that number of RDN attributes in the subject field SHALL NOT exceed five:

```
subject = [[
Ê   matter-noc-cat   = 0xABCD0004U,
Ê   matter-node-id   = 0x0102030405060708U,
Ê   matter-noc-cat   = 0xABCE0018U,
Ê   matter-fabric-id = 0xFAB000000000001DU,
Ê   matter-noc-cat   = 0xABCF0002U
]]
```

The following example illustrates an illegal subject DN due to the presence of the same CASE Authenticated Tag value with two different version numbers.

```
subject = [[
Ê   matter-node-id   = 0x0102030405060708U,
Ê   matter-fabric-id = 0xFAB000000000001DU,
Ê   matter-noc-cat   = 0xABCD0004U,    # <-- Value 0xABCD, Version 0x0004
Ê   matter-noc-cat   = 0xABCD0002U,    # <-- Value 0xABCD, Version 0x0002
]]
```

The following is an example of subject DN encoding for a Matter Root CA certificate. In this case, the Matter Root CA certificate is not associated with a specific Matter fabric:

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

```
subject = [[
Ê   matter-rcac-id = 0xCA0000000000001DU
]]
```

The following is another example of subject DN encoding for a Matter Root CA certificate . In this case, the Matter Root CA certificate is associated with a specific Matter fabric. This DN also encodes an issuer-specific common-name RDN attribute:

```
subject = [[
Ê   matter-rcac-id = 0xCA0000000000001DU,
Ê   matter-fabric-id  = 0xFAB000000000001DU,
Ê   common-name      = "ROOT CA HOME 3"
]]
```

6.5.7. Validity

The context-specific tags not-before [4] and not-after [5] SHALL be used to identify the not-before and not-after fields in the Matter certificate structure, which indicate the period of validity for the certificate. These two fields SHALL be encoded as unsigned integers. The value of these fields SHALL be encoded as a UTC time of type epoch-s (Epoch Time in Seconds).

Special value 0, when encoded in the not-after field, corresponds to the X.509/ RFCÊ5280defined special time value 99991231235959Z meaning no well-defined expiration date.

6.5.8. Public Key Algorithm

The context-specific tag pub-key-algo [7] SHALL be used to identify the public key algorithm field in the Matter certificate structure.

```
public-key-algorithm => UNSIGNED INTEGER [ range 8-bits ]
{
Ê   ec-pub-key  = 1,
}
```

The following values SHALL be defined for public-key-algorithm :

Table 59. Public Key Algorithm Object Identifiers

Value	ASN.1 OID
1	iso(1) member-body(2) us(840) ansi-x962(10045) keyType(2) ecPublicKey(1)

6.5.9. EC Curve Identifier

The context-specific tag ec-curve-id [8] SHALL be used to identify the elliptic curve field in the Matter certificate structure.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

```
elliptic-curve-id => UNSIGNED INTEGER [ range 8-bits ]
{
Ê   prime256v1  = 1,
}
```

The following values SHALL be defined for elliptic-curve-id :

Table 60. Elliptic Curve Object Identifiers

Value	ASN.1 OID
1	iso(1) member_body(2) us(840) ansi-x962(10045) curves(3) prime(1) prime256v1(7)

6.5.10. Public Key

The context-specific tag ec-pub-key [9] SHALL be used to identify the elliptic curve public key material field in the Matter certificate structure. The public key SHALL be a byte string representation of an uncompressed elliptic curve point as defined in section 2.3.3 of SEC1.

6.5.11. Extensions

The context-specific tag extensions [10] SHALL be used to identify the extensions field in the Matter certificate structure. The extensions list SHALL NOT contain more than one instance of a particular extension. The following table summarizes context-specific tags defined for the certificate extension types used in Matter.

```
extension => CHOICE OF
{
Ê   basic-cnstr [1]            : basic-constraints,
Ê   key-usage [2]              : UNSIGNED INTEGER [ range 16-bits ],
Ê   extended-key-usage [3]     : ARRAY [ length 1.. ] OF key-purpose-id,
Ê   subject-key-id [4]         : OCTET STRING [ length 20 ],
Ê   authority-key-id [5]       : OCTET STRING [ length 20 ],
Ê   future-extension [6]       : OCTET STRING,
}
```

Table 61. Extensions Object Identifiers

Tag	Matter Name	ASN.1 OID
1	basic-cnstr	joint-iso-itu-t(2) ds(5) certificateExtension(29) basicConstraints(19)
2	key-usage	joint-iso-itu-t(2) ds(5) certificateExtension(29) keyUsage(15)
3	extended-key-usage	joint-iso-itu-t(2) ds(5) certificateExtension(29) extendedKeyUsage(37)
4	subject-key-id	joint-iso-itu-t(2) ds(5) certificateExtension(29) subjectKeyIdentifier(14)

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Tag	Matter Name	ASN.1 OID
5	authority-key-id	joint-iso-itu-t(2) ds(5) certificateExtension(29) authorityKeyI! dentifier(35)
6	future-extension	any valid ASN.1 OID (future extension)

These context-specific tags identify the extension entries in the extensions list. The type of each extension is further described in the subsections below.

6.5.11.1. Basic Constraints Extension

The basic constraints extension identifies whether the subject of the certificate is a CA and the max! imum depth of valid certification paths that include this certificate.

When present, the basic constraints extension SHALL be treated as critical and it SHALL be marked as critical in the corresponding X.509 certificate. The critical field SHALL NOT be encoded in the Matter certificate structure.

The context-specific tag basic-cnstr [1] SHALL be used to identify a basic constraints extension entry in the Matter certificate extensions list.

```
basic-constraints => STRUCTURE [tag-order]
{
Ê  is-ca [1]                    : BOOLEAN,
Ê  path-len-constraint [2, optional] : UNSIGNED INTEGER [ range 8-bits ],
}
```

The is-ca field SHALL be encoded regardless of the value (true or false). The path-len-constraint MAY be present only when is-ca == true .

6.5.11.2. Key Usage Extension

The key usage extension defines the purpose of the key contained in the certificate.

When present, the key usage extension SHALL be treated as critical and it SHALL be marked as critical in the corresponding X.509 certificate. The critical field SHALL NOT be encoded in the Matter certificate structure.

The context-specific tag number key-usage [2] SHALL be used to identify a key usage extension entry in the Matter certificate extensions list.

The key-usage field is derived as a logical OR of all key-usage-flag values that apply to the corre! sponding public key:

```
key-usage-flag => UNSIGNED INTEGER [ range 16-bits ]
{
Ê  digitalSignature   = 0x0001,
Ê  nonRepudiation     = 0x0002,
Ê  keyEncipherment    = 0x0004,
```

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

```
Ê  dataEncipherment    = 0x0008,
Ê  keyAgreement        = 0x0010,
Ê  keyCertSign         = 0x0020,
Ê  CRLSign             = 0x0040,
Ê  encipherOnly        = 0x0080,
Ê  decipherOnly        = 0x0100,
}
```

6.5.11.3. Extended Key Usage Extension

The extended key usage extension indicates one or more purposes for which the certified public key may be used, in addition to or in place of the basic purposes indicated in the key usage exten! sion.

When present, the extended key usage extension SHALL be treated as critical and it SHALL be marked as critical in the corresponding X.509 certificate. The critical field SHALL NOT be encoded in the Matter certificate structure.

The context-specific tag number extended-key-usage [3] SHALL be used to identify an extended key usage extension entry in the Matter certificate extensions list.

The extended-key-usage field SHALL be encoded as an array of key-purpose-id values, where each key-purpose-id value SHALL be encoded as 8-bit unsigned integer:

```
key-purpose-id => UNSIGNED INTEGER [ range 8-bits ]
```

The following values SHALL be defined for key-purpose-id :

Table 62. Key Purpose Object Identifiers

Value	ASN.1 OID
1	iso(1), organization(3), dod(6), internet(1), security(5), mechanisms(5), pkix(7), 3, server! Auth(1)
2	iso(1), organization(3), dod(6), internet(1), security(5), mechanisms(5), pkix(7), 3, clien! tAuth(2)
3	iso(1), organization(3), dod(6), internet(1), security(5), mechanisms(5), pkix(7), 3, code! Signing(3)
4	iso(1), organization(3), dod(6), internet(1), security(5), mechanisms(5), pkix(7), 3, email! Protection(4)
5	iso(1), organization(3), dod(6), internet(1), security(5), mechanisms(5), pkix(7), 3, timeStamping(8)
6	iso(1), organization(3), dod(6), internet(1), security(5), mechanisms(5), pkix(7), 3, OCSP! Signing(9)

The key-purpose-id values in the extended-key-usage array SHALL be encoded in the same order as they appeared in the corresponding X.509 certificate.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

6.5.11.4. Subject Key Identifier Extension

The subject key identifier extension provides a means of identifying Matter certificates that contain a particular public key.

When present, the subject key identifier extension SHALL be treated as non-critical and it SHALL be marked as non-critical in the corresponding X.509 certificate. The critical field SHALL NOT be encoded in the Matter certificate structure.

The context-specific tag number subject-key-id [4] SHALL be used to identify a subject key identifier extension entry in the Matter certificate extensions list.

The Subject Key Identifier field SHALL be derived from the public key using method (1) described in section 4.2.1.2 of [RFC5280]. Thus, the subject-key-id SHALL be composed of the 160-bit SHA-1 hash of the certificateÕs subject public key value.

See Section 6.1.2, ÒKey Identifier Extension ConstraintsÓ for additional constraints.

6.5.11.5. Authority Key Identifier Extension

The authority key identifier extension provides a means of identifying the public key corresponding to the private key used to sign a Matter certificate.

When present, the authority key identifier extension SHALL be treated as non-critical and it SHALL be marked as non-critical in the corresponding X.509 certificate. The critical field SHALL NOT be encoded in the Matter certificate structure.

The context-specific tag number authority-key-id [5] SHALL be used to identify an authority key identifier extension entry in the Matter certificate extensions list.

Note that the authority key identifier extension field in an X.509 certificate may optionally include issuer and serial number fields, which are not supported by Matter certificates.

The Authority Key Identifier field SHALL be derived from the public key using method (1) described in section 4.2.1.2 of [RFC5280]. Thus, the authority-key-id SHALL be composed of the 160-bit SHA-1 hash of the public key used to verify the certificateÕs signature.

See Section 6.1.2, ÒKey Identifier Extension ConstraintsÓ for additional constraints.

6.5.11.6. Future Extension

The Matter certificate is designed with extensibility in mind and this field is added to support arbitrary certificate extension in the future.

Note that implementations that do not support specific future extension will ignore it but will be able to use it for the Matter certificate signature validation. If ignored extension is marked as critical then validation of the corresponding Matter certificate SHALL fail.

The context-specific tag number future-extension [6] SHALL be used to identify all future extension entries in the Matter certificate extensions list. There MAY be more than one future extension field.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

The future-extension field SHALL be encoded as OCTET STRING and it SHALL be an exact copy of the DER encoded extension field (including the DER encoded ASN.1 OID of the extension) in the cor! responding X.509 certificate. These extension fields in a Matter certificate SHALL be encoded in the same order as they appeared in the original X.509 certificate.

6.5.12. Matter certificate Extensions Encoding Rules

The rules that SHALL be followed when encoding the Matter certificate are:

- For a Matter Node Operational Certificate (NOC) :
  - The basic constraints extension SHALL be encoded with is-ca set to false .
  - The key usage extension SHALL be encoded with exactly one flag: digitalSignature .
  - The extended key usage extension SHALL be encoded with exactly two key-purpose-id val! ues: serverAuth and clientAuth .
  - The subject key identifier extension SHALL be present.
  - The authority key identifier extension SHALL be present.
- For Matter ICA Certificate and Matter Root CA Certificate :
  - The basic constraints extension SHALL be encoded with is-ca set to true .
  - The key usage extension SHALL be encoded with exactly two flags: keyCertSign and CRLSign
  - The extended key usage extension SHALL NOT be present.
  - The subject key identifier extension SHALL be present.
  - The authority key identifier extension SHALL be present.
  - For the Matter Root CA Certificate the authority key identifier extension SHALL be equal to the subject key identifier extension.
- For a Matter Firmware Signing Certificate these rules SHALL be followed:
  - The basic constraints extension SHALL be encoded with is-ca set to false .
  - The key usage extension SHALL be encoded with exactly one flag: digitalSignature .
  - The extended key usage extension SHALL be encoded with exactly one key-purpose-id val! ues: codeSigning.
  - The subject key identifier extension SHALL be present.
  - The authority key identifier extension SHALL be present.
- The extensions SHALL appear in the same order in the Matter certificate and in the correspond! ing X.509 certificates.

Note that Matter doesnÕt specify how firmware images are signed and implementation of firmware image signing is manufacturer-specific. However, since firmware image signing is a common fea! ture, the format for Matter TLV certificates has affordances for encoding firmware signing certifi! cates.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

6.5.13. Signature

The context-specific tag signature [11] SHALL be used to identify the signature field in the Matter certificate structure.

An ec-signature is the encoding of the signature as defined in Section 3.5.3, ÒSignature and verifica! tionÓ.

```
ec-signature => OCTET STRING [ length ( CRYPTO_GROUP_SIZE_BYTES * 2) ]
```

6.5.14. Invalid Matter certificates

The Matter certificate is considered invalid if it violates Matter certificate encoding rules defined in this section. The processing of invalid Matter certificate SHOULD fail and an error SHOULD be reported to the application. Here is a non-exhaustive list of errors that may invalidate the certifi! cate:

- ¥ Matter certificate structure includes elements that are not defined in this section.
- ¥ Matter certificate elements are encoded in a wrong order.
- ¥ Matter certificate element has wrong type.
- ¥ Length of an element is outside of its valid range, for example:
  - %serial-num field is longer than 20 octets.
  - %not-before or not-after field is longer than 32-bits.
  - %subject-key-id or authority-key-id field is different from 20 octets.
- ¥ Matter OID values encoded in Matter certificate are not defined in this section, for example:
  - %sig-algo field encodes value, which is not defined in Table 55, ÒSignature Algorithm Object IdentifiersÓ .
  - %pub-key-algo field encodes value, which is not defined in Table 59, ÒPublic Key Algorithm Object IdentifiersÓ .
  - %ec-curve-id field encodes value, which is not defined in Table 60, ÒElliptic Curve Object Iden! tifiersÓ.
  - %key-purpose-id field of the Extended Key Usage Extension encodes value, which is not defined in Table 62, ÒKey Purpose Object IdentifiersÓ .
- ¥ Invalid Matter Distinguished Names encoding for Issuer and Subject DNs. Refer to Section 6.5.6.3, ÒMatter DN Encoding RulesÓ for more details. For example:
  - %Node Subject DN doesnÕt include Matter specific matter-node-id or matter-fabric-id attribute.
  - %Firmware signing Subject DN doesnÕt include Matter specific matter-firmware-signing-id attribute.
  - %Intermediate CA Subject DN doesnÕt include Matter specific matter-icac-id attribute.
  - %Root CA Subject DN doesnÕt include Matter specific matter-rcac-id attribute.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

- %Certificate Subject DN encodes matter-node-id and matter-rcac-id , which contradict each other.
- %Multiple matter-cat-id with the same identifier value and different version numbers, or any matter-cat-id with a version number of 0.

¥ CA certificate doesnÕt have Basic Constraints Extension is-ca field set to 'true'.

¥ key-usage field of the Key Usage Extension has undefined flags.

¥ Certificate extension that SHALL be marked as critical is marked as non-critical in the X.509 representation and vise versa.

6.5.15. Examples

6.5.15.1. Example of Operational Root CA Certificate (RCAC)

The same RCAC in X.509 and Matter TLV formats is presented in this section.

RCAC with Corresponding Private Key in an X.509 PEM Format

```
-----BEGIN CERTIFICATE-----
MIIBnTCCAUOgAwIBAgIIWeqmMpR/VBwwCgYIKoZIzj0EAwIwIjEgMB4GCisGAQQB
gqJ8AQQMEENBQ0FDQUNBMDAwMDAwMDEwHhcNMjAxMDE1MTQyMzQzWhcNNDAxMDE1
MTQyMzQyWjAiMSAwHgYKKwYBBAGConwBBAwQQ0FDQUNBQ0EwMDAwMDAwMTBZMBMG
ByqGSM49AgEGCCqGSM49AwEHA0IABBNTo7PvHacIxJCASAFOQH1ZkM4ivE6zPppa
yyWoVgPrptzYITZmpORPWsoT63Z/r6fc3dwzQR+CowtUPdHSS6ijYzBhMA8GA1Ud
EwEB/wQFMAMBAf8wDgYDVR0PAQH/BAQDAgEGMB0GA1UdDgQWBBQTr4GrNzdLLtKp
ZJsSt6OkKH4VHTAfBgNVHSMEGDAWgBQTr4GrNzdLLtKpZJsSt6OkKH4VHTAKBggq
hkjOPQQDAgNIADBFAiBFgWRGbI8ZWrwKu3xstaJ6g/QdN/jVO+7FIKvSoNoFCQIh
ALinwlwELjDPZNww/jNOEgAZZk5RUEkTT1eBI4RE/HUx
-----END CERTIFICATE-----

-----BEGIN EC PRIVATE KEY-----
MHcCAQEEIH1zW+/pFqHAygL4ypiB5CZjqq+aucQzsom+JnAQdXQaoAoGCCqGSM49
AwEHoUQDQgAEE1Ojs+8dpwjEkIBIAU5AfVmQziK8TrM+mlrLJahWA+um3NghNmak
5E9ayhPrdn+vp9zd3DNBH4KjC1Q90dJLqA==
-----END EC PRIVATE KEY-----
```

RCAC Printed in an X.509 DER Format

```
Certificate:
Ê   Data:
Ê      Version: 3 (0x2)
Ê      Serial Number: 6479173750095827996 (0x59eaa632947f541c)
Ê      Signature Algorithm: ecdsa-with-SHA256
Ê      Issuer: 1.3.6.1.4.1.37244.1.4 = CACACACA00000001
Ê      Validity
Ê         Not Before: Oct 15 14:23:43 2020 GMT
Ê         Not After : Oct 15 14:23:42 2040 GMT
Ê      Subject: 1.3.6.1.4.1.37244.1.4 = CACACACA00000001
Ê      Subject Public Key Info:
```


Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

```
Ê         Public Key Algorithm: id-ecPublicKey
Ê            Public-Key: (256 bit)
Ê            pub:
Ê               04:13:53:a3:b3:ef:1d:a7:08:c4:90:80:48:01:4e:
Ê               40:7d:59:90:ce:22:bc:4e:b3:3e:9a:5a:cb:25:a8:
Ê               56:03:eb:a6:dc:d8:21:36:66:a4:e4:4f:5a:ca:13:
Ê               eb:76:7f:af:a7:dc:dd:dc:33:41:1f:82:a3:0b:54:
Ê               3d:d1:d2:4b:a8
Ê            ASN1 OID: prime256v1
Ê            NIST CURVE: P-256
Ê      X509v3 extensions:
Ê         X509v3 Basic Constraints: critical
Ê            CA:TRUE
Ê         X509v3 Key Usage: critical
Ê            Certificate Sign, CRL Sign
Ê         X509v3 Subject Key Identifier:
Ê            13:AF:81:AB:37:37:4B:2E:D2:A9:64:9B:12:B7:A3:A4:28:7E:15:1D
Ê         X509v3 Authority Key Identifier:
Ê            keyid:13:AF:81:AB:37:37:4B:2E:D2:A9:64:9B:12:B7:A3:A4:28:7E:15:1D

Ê   Signature Algorithm: ecdsa-with-SHA256
Ê        30:45:02:20:45:81:64:46:6c:8f:19:5a:bc:0a:bb:7c:6c:b5:
Ê        a2:7a:83:f4:1d:37:f8:d5:3b:ee:c5:20:ab:d2:a0:da:05:09:
Ê        02:21:00:b8:a7:c2:5c:04:2e:30:cf:64:dc:30:fe:33:4e:12:
Ê        00:19:66:4e:51:50:49:13:4f:57:81:23:84:44:fc:75:31
```

RCAC in Matter TLV Format

```
15 30 01 08 59 ea a6 32 94 7f 54 1c 24 02 01 37 03 27 14 01 00 00 00 ca
ca ca ca 18 26 04 ef 17 1b 27 26 05 6e b5 b9 4c 37 06 27 14 01 00 00 00
ca ca ca ca 18 24 07 01 24 08 01 30 09 41 04 13 53 a3 b3 ef 1d a7 08 c4
90 80 48 01 4e 40 7d 59 90 ce 22 bc 4e b3 3e 9a 5a cb 25 a8 56 03 eb a6
dc d8 21 36 66 a4 e4 4f 5a ca 13 eb 76 7f af a7 dc dd dc 33 41 1f 82 a3
0b 54 3d d1 d2 4b a8 37 0a 35 01 29 01 18 24 02 60 30 04 14 13 af 81 ab
37 37 4b 2e d2 a9 64 9b 12 b7 a3 a4 28 7e 15 1d 30 05 14 13 af 81 ab 37
37 4b 2e d2 a9 64 9b 12 b7 a3 a4 28 7e 15 1d 18 30 0b 40 45 81 64 46 6c
8f 19 5a bc 0a bb 7c 6c b5 a2 7a 83 f4 1d 37 f8 d5 3b ee c5 20 ab d2 a0
da 05 09 b8 a7 c2 5c 04 2e 30 cf 64 dc 30 fe 33 4e 12 00 19 66 4e 51 50
49 13 4f 57 81 23 84 44 fc 75 31 18
```

RCAC Printed in Matter TLV Schema Format

```
matter-certificate = {
Ê   serial-num        = 59 EA A6 32 94 7F 54 1C,
Ê   sig-algo          = 0x01U,
Ê   issuer            = [[ matter-rcac-id = 0xCACACACA00000001U ]],
Ê   not-before        = 0x271B17EFU,
Ê   not-after         = 0x4CB9B56EU,
Ê   subject           = [[ matter-rcac-id = 0xCACACACA00000001U ]],
Ê   pub-key-algo      = 0x01U,
```

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

```
Ê  ec-curve-id       = 0x01U,
Ê  ec-pub-key        = 04 13 53 A3 B3 EF 1D A7 08 C4 90 80 48 01 4E 40
Ê                    7D 59 90 CE 22 BC 4E B3 3E 9A 5A CB 25 A8 56 03
Ê                    EB A6 DC D8 21 36 66 A4 E4 4F 5A CA 13 EB 76 7F
Ê                    AF A7 DC DD DC 33 41 1F 82 A3 0B 54 3D D1 D2 4B
Ê                    A8,
Ê  extensions        = [[
Ê     basic-constraints  = {
Ê        is-ca = true
Ê     },
Ê     key-usage          = 0x60U,
Ê     subject-key-id     = 13 AF 81 AB 37 37 4B 2E D2 A9 64 9B 12 B7 A3 A4
Ê                          28 7E 15 1D,
Ê     authority-key-id   = 13 AF 81 AB 37 37 4B 2E D2 A9 64 9B 12 B7 A3 A4
Ê                          28 7E 15 1D,
Ê  ]],
Ê  signature         = 45 81 64 46 6C 8F 19 5A BC 0A BB 7C 6C B5 A2 7A
Ê                    83 F4 1D 37 F8 D5 3B EE C5 20 AB D2 A0 DA 05 09
Ê                    B8 A7 C2 5C 04 2E 30 CF 64 DC 30 FE 33 4E 12 00
Ê                    19 66 4E 51 50 49 13 4F 57 81 23 84 44 FC 75 31
}
```

6.5.15.2. Example of Operational Intermediate CA Certificate (ICAC)

The same ICAC in X.509 and Matter TLV formats is presented in this section. An issuer of this ICAC is RCAC from previous section.

ICAC with Corresponding Private Key in an X.509 PEM Format

```
-----BEGIN CERTIFICATE-----
MIIBnTCCAUOgAwIBAgIILbREhVZBrt8wCgYIKoZIzj0EAwIwIjEgMB4GCisGAQQB
gqJ8AQQMEENBQ0FDQUNBMDAwMDAwMDEwHhcNMjAxMDE1MTQyMzQzWhcNNDAxMDE1
MTQyMzQyWjAiMSAwHgYKKwYBBAGConwBAwwQQ0FDQUNBQ0EwMDAwMDAwMzBZMBMG
ByqGSM49AgEGCCqGSM49AwEHA0IABMXQhhu4+QxAXBIxTkxevuqTn3J3S8wzI54v
Wfb0avjcfUaCoOPMxkbm3ynqhr9WKucgqJgzfTg/MsCgnkFgGeqjYzBhMA8GA1Ud
EwEB/wQFMAMBAf8wDgYDVR0PAQH/BAQDAgEGMB0GA1UdDgQWBBRTUtcFnpwVpQiQ
aGKGSAGinx9B0zAfBgNVHSMEGDAWgBQTr4GrNzdLLtKpZJsSt6OkKH4VHTAKBggq
hkjOPQQDAgNIADBFAiEAhBoG1Dten+zSToexJE61HGos8g2bXmugfxHmAC9+DKMC
IE4ypgLDYJ0AktNIvb0ZihFGRr1BzxA3g2Qa4l4/I/0m
-----END CERTIFICATE-----

-----BEGIN EC PRIVATE KEY-----
MHcCAQEEIBGEO9zwrSBtsQJRpU2sWB11+ZL8tSJ1KiFs15xxdUapoAoGCCqGSM49
AwEHoUQDQgAExdCGG7j5DEBcEjFOTF6+6pOfcndLzDMjni9Z9vRq+Nx9RoKg48zG
RubfKeqGv1Yq5yComDN9OD8ywKCeQWAZ6g==
-----END EC PRIVATE KEY-----
```

ICAC Printed in an X.509 DER Format

```
Certificate:
```


Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

```
Ê  Data:
Ê      Version: 3 (0x2)
Ê      Serial Number: 3293332566983159519 (0x2db444855641aedf)
Ê      Signature Algorithm: ecdsa-with-SHA256
Ê      Issuer: 1.3.6.1.4.1.37244.1.4 = CACACACA00000001
Ê      Validity
Ê          Not Before: Oct 15 14:23:43 2020 GMT
Ê          Not After : Oct 15 14:23:42 2040 GMT
Ê      Subject: 1.3.6.1.4.1.37244.1.3 = CACACACA00000003
Ê      Subject Public Key Info:
Ê          Public Key Algorithm: id-ecPublicKey
Ê              Public-Key: (256 bit)
Ê              pub:
Ê                  04:c5:d0:86:1b:b8:f9:0c:40:5c:12:31:4e:4c:5e:
Ê                  be:ea:93:9f:72:77:4b:cc:33:23:9e:2f:59:f6:f4:
Ê                  6a:f8:dc:7d:46:82:a0:e3:cc:c6:46:e6:df:29:ea:
Ê                  86:bf:56:2a:e7:20:a8:98:33:7d:38:3f:32:c0:a0:
Ê                  9e:41:60:19:ea
Ê              ASN1 OID: prime256v1
Ê              NIST CURVE: P-256
Ê      X509v3 extensions:
Ê          X509v3 Basic Constraints: critical
Ê              CA:TRUE
Ê          X509v3 Key Usage: critical
Ê              Certificate Sign, CRL Sign
Ê          X509v3 Subject Key Identifier:
Ê              53:52:D7:05:9E:9C:15:A5:08:90:68:62:86:48:01:A2:9F:1F:41:D3
Ê          X509v3 Authority Key Identifier:
Ê              keyid:13:AF:81:AB:37:37:4B:2E:D2:A9:64:9B:12:B7:A3:A4:28:7E:15:1D

Ê  Signature Algorithm: ecdsa-with-SHA256
Ê       30:45:02:21:00:84:1a:06:d4:3b:5e:9f:ec:d2:4e:87:b1:24:
Ê       4e:b5:1c:6a:2c:f2:0d:9b:5e:6b:a0:7f:11:e6:00:2f:7e:0c:
Ê       a3:02:20:4e:32:a6:02:c3:60:9d:00:92:d3:48:bd:bd:19:8a:
Ê       11:46:46:bd:41:cf:10:37:83:64:1a:e2:5e:3f:23:fd:26
```

ICAC in Matter TLV Format

```
15 30 01 08 2d b4 44 85 56 41 ae df 24 02 01 37 03 27 14 01 00 00 00 ca
ca ca ca 18 26 04 ef 17 1b 27 26 05 6e b5 b9 4c 37 06 27 13 03 00 00 00
ca ca ca ca 18 24 07 01 24 08 01 30 09 41 04 c5 d0 86 1b b8 f9 0c 40 5c
12 31 4e 4c 5e be ea 93 9f 72 77 4b cc 33 23 9e 2f 59 f6 f4 6a f8 dc 7d
46 82 a0 e3 cc c6 46 e6 df 29 ea 86 bf 56 2a e7 20 a8 98 33 7d 38 3f 32
c0 a0 9e 41 60 19 ea 37 0a 35 01 29 01 18 24 02 60 30 04 14 53 52 d7 05
9e 9c 15 a5 08 90 68 62 86 48 01 a2 9f 1f 41 d3 30 05 14 13 af 81 ab 37
37 4b 2e d2 a9 64 9b 12 b7 a3 a4 28 7e 15 1d 18 30 0b 40 84 1a 06 d4 3b
5e 9f ec d2 4e 87 b1 24 4e b5 1c 6a 2c f2 0d 9b 5e 6b a0 7f 11 e6 00 2f
7e 0c a3 4e 32 a6 02 c3 60 9d 00 92 d3 48 bd bd 19 8a 11 46 46 bd 41 cf
10 37 83 64 1a e2 5e 3f 23 fd 26 18
```

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

ICAC Printed in Matter TLV Schema Format

```
matter-certificate = {
Ê   serial-num        = 2D B4 44 85 56 41 AE DF,
Ê   sig-algo          = 0x01U,
Ê   issuer            = [[ matter-rcac-id = 0xCACACACA00000001U ]],
Ê   not-before        = 0x271B17EFU,
Ê   not-after         = 0x4CB9B56EU,
Ê   subject           = [[ matter-icac-id = 0xCACACACA00000003U ]],
Ê   pub-key-algo      = 0x01U,
Ê   ec-curve-id       = 0x01U,
Ê   ec-pub-key        = 04 C5 D0 86 1B B8 F9 0C 40 5C 12 31 4E 4C 5E BE
Ê                     EA 93 9F 72 77 4B CC 33 23 9E 2F 59 F6 F4 6A F8
Ê                     DC 7D 46 82 A0 E3 CC C6 46 E6 DF 29 EA 86 BF 56
Ê                     2A E7 20 A8 98 33 7D 38 3F 32 C0 A0 9E 41 60 19
Ê                     EA,
Ê   extensions        = [[
Ê      basic-constraints  = {
Ê         is-ca = true
Ê      },
Ê      key-usage          = 0x60U,
Ê      subject-key-id     = 53 52 D7 05 9E 9C 15 A5 08 90 68 62 86 48 01 A2
Ê                           9F 1F 41 D3,
Ê      authority-key-id   = 13 AF 81 AB 37 37 4B 2E D2 A9 64 9B 12 B7 A3 A4
Ê                           28 7E 15 1D,
Ê   ]],
Ê   signature         = 84 1A 06 D4 3B 5E 9F EC D2 4E 87 B1 24 4E B5 1C
Ê                     6A 2C F2 0D 9B 5E 6B A0 7F 11 E6 00 2F 7E 0C A3
Ê                     4E 32 A6 02 C3 60 9D 00 92 D3 48 BD BD 19 8A 11
Ê                     46 46 BD 41 CF 10 37 83 64 1A E2 5E 3F 23 FD 26
}
```

6.5.15.3. Example of Node Operational Certificate (NOC)

The same NOC in X.509 and Matter TLV formats is presented in this section. An issuer of this NOC is ICAC from previous section.

NOC with Corresponding Private Key in an X.509 PEM Format

```
-----BEGIN CERTIFICATE-----
MIIB4DCCAYagAwIBAgIIPvz/FwK5oXowCgYIKoZIzj0EAwIwIjEgMB4GCisGAQQB
gqJ8AQMMEENBQ0FDQUNBMDAwMDAwMDMwHhcNMjAxMDE1MTQyMzQzWhcNNDAxMDE1
MTQyMzQyWjBEMSAwHgYKKwYBBAGConwBAQwQREVERURFREUwMDAxMDAwMTEgMB4G
CisGAQQBgqJ8AQUMEEZBQjAwMDAwMDAwMDAwMUQwWTATBgcqhkjOPQIBBggqhkjO
PQMBBwNCAASaKiFvs53WtvohG4NciePmr7ZsFPdYMZVPn/T3o/ARLIoNjq8pxlMp
TUju4HCKAyzKOTk8OntG8YGuoHj+rYODo4GDMIGAMAwGA1UdEwEB/wQCMAAwDgYD
VR0PAQH/BAQDAgeAMCAGA1UdJQEB/wQWMBQGCCsGAQUFBwMCBggrBgEFBQcDATAd
BgNVHQ4EFgQUn1Wia35DA+YIg+kTv5T0+14qYWEwHwYDVR0jBBgwFoAUU1LXBZ6c
FaUIkGhihkgBop8fQdMwCgYIKoZIzj0EAwIDSAAwRQIgeVXCAmMLS6TVkSUmMi/f
KPie3+WvnA5XK9ihSqq7TRICIQC4PKF8ewX7Fkt315xSlhMxa8/ReJXksqTyQEuY
```


Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

```
FzJxWQ==
-----END CERTIFICATE-----

-----BEGIN EC PRIVATE KEY-----
MHcCAQEEIKVls/ooqO1qdPtvD/ik00DZ4a6Y8h36HwpZpOoCGhYnoAoGCCqGSM49
AwEHoUQDQgAEmiohb7Od1rb6IRuDXInj5q+2bBT3WDGVT5/096PwESyKDY6vKcZT
KU1I7uBwigMsyjk5PDp7RvGBrqB4/q2Dgw==
-----END EC PRIVATE KEY-----
```

NOC Printed in an X.509 DER Format

```
Certificate:
Ê   Data:
Ê       Version: 3 (0x2)
Ê       Serial Number: 4538782998777667962 (0x3efcff1702b9a17a)
Ê       Signature Algorithm: ecdsa-with-SHA256
Ê       Issuer: 1.3.6.1.4.1.37244.1.3 = CACACACA00000003
Ê       Validity
Ê           Not Before: Oct 15 14:23:43 2020 GMT
Ê           Not After : Oct 15 14:23:42 2040 GMT
Ê       Subject: 1.3.6.1.4.1.37244.1.1 = DEDEDEDE00010001, 1.3.6.1.4.1.37244.1.5 =
FAB000000000001D
Ê       Subject Public Key Info:
Ê           Public Key Algorithm: id-ecPublicKey
Ê               Public-Key: (256 bit)
Ê               pub:
Ê                   04:9a:2a:21:6f:b3:9d:d6:b6:fa:21:1b:83:5c:89:
Ê                   e3:e6:af:b6:6c:14:f7:58:31:95:4f:9f:f4:f7:a3:
Ê                   f0:11:2c:8a:0d:8e:af:29:c6:53:29:4d:48:ee:e0:
Ê                   70:8a:03:2c:ca:39:39:3c:3a:7b:46:f1:81:ae:a0:
Ê                   78:fe:ad:83:83
Ê               ASN1 OID: prime256v1
Ê               NIST CURVE: P-256
Ê       X509v3 extensions:
Ê           X509v3 Basic Constraints: critical
Ê               CA:FALSE
Ê           X509v3 Key Usage: critical
Ê               Digital Signature
Ê           X509v3 Extended Key Usage: critical
Ê               TLS Web Client Authentication, TLS Web Server Authentication
Ê           X509v3 Subject Key Identifier:
Ê               9F:55:A2:6B:7E:43:03:E6:08:83:E9:13:BF:94:F4:FB:5E:2A:61:61
Ê           X509v3 Authority Key Identifier:
Ê               keyid:53:52:D7:05:9E:9C:15:A5:08:90:68:62:86:48:01:A2:9F:1F:41:D3

Ê   Signature Algorithm: ecdsa-with-SHA256
Ê        30:45:02:20:79:55:c2:02:63:0b:4b:a4:d5:91:25:26:32:2f:
Ê        df:28:f8:9e:df:e5:af:9c:0e:57:2b:d8:a1:4a:aa:bb:4d:12:
Ê        02:21:00:b8:3c:a1:7c:7b:05:fb:16:4b:77:d7:9c:52:96:13:
Ê        31:6b:cf:d1:78:95:e4:b2:a4:f2:40:4b:98:17:32:71:59
```

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

NOC in Matter TLV Format

```
15 30 01 08 3e fc ff 17 02 b9 a1 7a 24 02 01 37 03 27 13 03 00 00 00 ca
ca ca ca 18 26 04 ef 17 1b 27 26 05 6e b5 b9 4c 37 06 27 11 01 00 01 00
de de de de 27 15 1d 00 00 00 00 00 b0 fa 18 24 07 01 24 08 01 30 09 41
04 9a 2a 21 6f b3 9d d6 b6 fa 21 1b 83 5c 89 e3 e6 af b6 6c 14 f7 58 31
95 4f 9f f4 f7 a3 f0 11 2c 8a 0d 8e af 29 c6 53 29 4d 48 ee e0 70 8a 03
2c ca 39 39 3c 3a 7b 46 f1 81 ae a0 78 fe ad 83 83 37 0a 35 01 28 01 18
24 02 01 36 03 04 02 04 01 18 30 04 14 9f 55 a2 6b 7e 43 03 e6 08 83 e9
13 bf 94 f4 fb 5e 2a 61 61 30 05 14 53 52 d7 05 9e 9c 15 a5 08 90 68 62
86 48 01 a2 9f 1f 41 d3 18 30 0b 40 79 55 c2 02 63 0b 4b a4 d5 91 25 26
32 2f df 28 f8 9e df e5 af 9c 0e 57 2b d8 a1 4a aa bb 4d 12 b8 3c a1 7c
7b 05 fb 16 4b 77 d7 9c 52 96 13 31 6b cf d1 78 95 e4 b2 a4 f2 40 4b 98
17 32 71 59 18
```

NOC Printed in Matter TLV Schema Format

```
matter-certificate = {
Ê   serial-num         = 3E FC FF 17 02 B9 A1 7A,
Ê   sig-algo           = 0x01U,
Ê   issuer             = [[ matter-icac-id = 0xCACACACA00000003U ]],
Ê   not-before         = 0x271B17EFU,
Ê   not-after          = 0x4CB9B56EU,
Ê   subject            = [[ matter-node-id = 0xDEDEDEDE00010001U,
Ê                           matter-fabric-id = 0xFAB000000000001DU ]],
Ê   pub-key-algo       = 0x01U,
Ê   ec-curve-id        = 0x01U,
Ê   ec-pub-key         = 04 9A 2A 21 6F B3 9D D6 B6 FA 21 1B 83 5C 89 E3
Ê                        E6 AF B6 6C 14 F7 58 31 95 4F 9F F4 F7 A3 F0 11
Ê                        2C 8A 0D 8E AF 29 C6 53 29 4D 48 EE E0 70 8A 03
Ê                        2C CA 39 39 3C 3A 7B 46 F1 81 AE A0 78 FE AD 83
Ê                        83,
Ê   extensions         = [[
Ê       basic-constraints  = {
Ê           is-ca = false
Ê       },
Ê       key-usage          = 0x01U,
Ê       extended-key-usage = [ 0x02U, 0x01U ],
Ê       subject-key-id     = 9F 55 A2 6B 7E 43 03 E6 08 83 E9 13 BF 94 F4 FB
Ê                            5E 2A 61 61,
Ê       authority-key-id   = 53 52 D7 05 9E 9C 15 A5 08 90 68 62 86 48 01 A2
Ê                            9F 1F 41 D3,
Ê   ]],
Ê   signature          = 79 55 C2 02 63 0B 4B A4 D5 91 25 26 32 2F DF 28
Ê                        F8 9E DF E5 AF 9C 0E 57 2B D8 A1 4A AA BB 4D 12
Ê                        B8 3C A1 7C 7B 05 FB 16 4B 77 D7 9C 52 96 13 31
Ê                        6B CF D1 78 95 E4 B2 A4 F2 40 4B 98 17 32 71 59
}
```


Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

6.6. Access Control

6.6.1. Scope and Purpose

This section specifies the features related to controlling access to a NodeÕs Endpoint Clusters ("Tar! gets" hereafter) from other Nodes. The overall features are collectively named "Access Control" hereafter.

The Access Control features aim to ensure that only authorized Nodes are permitted access to given application-layer functionality exposed by the Data Model, through the Interaction Model. Access Control is the fundamental link between the Secure Channel and the Interaction Model.

In order to implement a policy of Access Control, Administrators on the fabric create and maintain a consistent distributed configuration of Access Control Lists (ACLs) across all Nodes. Each Node has an ACL containing Access Control Entries which codify the policy. The Access Control Cluster exposes a data model view of a NodeÕs ACL which enables its maintenance.

6.6.2. Model

The Access Control system is rule-based with no implicit access permitted by default. Access to a NodeÕs Targets is denied unless the Access Control system grants the required privilege level to a given Subject to interact with given Targets on that Node. Initial Access Control privileges are boot! strapped during the commissioning phase, and maintained thereafter by Administrators.

All access privileges, from the AccessControlEntryPrivilegeEnum enumeration, SHALL be granted by Access Control. Thus, the Initiator ("Subject" hereafter) of any Interaction Model action SHALL NOT succeed in executing a given action on a NodeÕs Target unless that NodeÕs Access Control sys! tem explicitly grants the required privilege to that Subject for that particular action.

The Access Control system grants privileges by checking and verifying all attempted access against rules explicitly codified in Access Control Entries within the NodeÕs Access Control List. Addition! ally, Access Control implicitly grants administrative access privileges to an Administrative Subject during a NodeÕs commissioning phase .

Access Control Entries contain:

¥ A FabricIndex scoping the entry to the Associated Fabric .

¥ A Privilege level granted by the entry (see AccessControlEntryPrivilegeEnum)

%View: reading or subscribing to data from a non- Proxy

%Proxy View : reading or subscribing to data from a Proxy

%Operate: writing operational data and invoking operational commands

%Manage: writing configuration data and invoking configuration commands (for example, Binding and GroupClusters access)

%Administer : writing administrative data and invoking administrative commands (for exam! ple, Access Control and CommissioningClusters access)

¥ A list of target Clusters to which the entry applies

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

- A list of source Subjects to which the entry applies
- An Authentication Mode that describes the type of secure channel authentication method to which the entryÕs subjects apply

6.6.2.1. Subjects

The meaning of a "Subject" is primarily that of describing the source for an action, using a given authentication method provided by the Secure Channel architecture. A subject SHALL be one of:

- A passcode, identified by a Passcode ID, authenticated locally by a PASE session, during the com! missioning phase.
- Note that any Passcode ID other than 0, which is the default commissioning passcode, is reserved for future use.
- Furthermore, ACL entries with a PASE authentication mode SHALL NOT be explicitly added to the Access Control List, since there is an invisible implicit administrative entry (see Section 6.6.2.8, ÒBootstrapping of the Access Control ClusterÓ) always equivalently present on the Com! missionee (but not the Commissioner) during PASE sessions.
- A source node, authenticated by a CASE session using its Operational Certificate, during the operational phase. The source node can be identified by its Node ID and/or by CASE Authenti! cated Tags.
- A destination group, identified by a destination Group ID in the message, authenticated by an Operational Group Key from the Group Key Management Cluster, during the operational phase.

6.6.2.1.1. PASE and Group Subjects

Note that the subject is not considered to be an individual Node when the authentication is via pass! code or group symmetric key; in these cases, the administrative root of trust is conditional only upon bearing the correct passcode during session establishment, or bearing the Operational Group Key when constructing a group message.

6.6.2.1.2. Subjects identified by CASE Authenticated Tag

In contrast, a CASE Authenticated Tag (CAT) is a special subject distinguished name within the Oper! ational Certificate shared during CASE session establishment that functions as a group-like tag. Such a tag can be applied to several Nodes, thereby facilitating management of Access Control Entries that use the same set of Nodes as subjects. Because these tags are authenticated within the CASE session context, the administrative root of trust does chain back through the individual source Node to the FabricÕs trusted root . This makes CATs suitable for group-like use while main! taining secure authentication and attribution ability.

Each CAT is 32-bit and equally divided into identifier value and its corresponding version:

- The upper 16-bits are allocated to identifier value
- The lower 16-bits are allocated to the version number

The version number represents current version of specific identifier value. An Administrator MAY increment the version number on any changes in the set of Nodes sharing the given tag. Version number is a monotonically increasing natural number in the range of 1 to 65535. A version number

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

of 0 is invalid and SHALL NOT be used. On reaching the maximum value (65535), wrap-around is not supported and the tag identifier SHOULD be retired by an Administrator since version increase will no longer be possible.

When a CAT appears in the Subjects list of an Access Control Entry, it SHALL be encoded within the CASE Authenticated Tag sub-space of Node Identifiers , with the upper 32 bits set to 0xFFFF_FFFD. Note that this encoding cannot appear as an operational Node ID. It is merely a sub-encoding allowing the 64-bit scalars in an Access Control EntryÕs Subjects list to represent both Node IDs and CATs.

Example CASE Authenticated Tags:

- ¥ Tag identifier 0xAB12, Version 0x0003
  - %CAT value: 0xAB12_0003
  - %Appears in Access Control Entry Subjects list as 0xFFFF_FFFD_AB12_0003
  - %Appears in Node Operational Certificate subject under OID 1.3.6.1.4.1.37244.1.6 with value 0xAB120003
    - & Would appear in X.509 certificate subject under OID 1.3.6.1.4.1.37244.1.6 as UTF8String AB120003 for signature validation purposes.
- ¥ Tag identifier 0x071C, Version 0x1074
  - %CAT value: 0x071C_1074
  - %Appears in Access Control Entry Subjects list as 0xFFFF_FFFD_071C_1074
  - %Appears in Node Operational Certificate subject under OID 1.3.6.1.4.1.37244.1.6 with value 0x071C1074
    - & Would appear in X.509 certificate subject under OID 1.3.6.1.4.1.37244.1.6 as UTF8String 071C1074 for signature validation purposes.

6.6.2.2. Wildcards

The Subjects list of an Access Control Entry MAY grant a given privilege to more than one Subject, if the Authentication Mode allows it, such as in the case of the CASE and Group Authentication Modes. An empty Subjects list SHALL mean that every possible Subject employing the stated Authentication Mode is granted the entryÕs privilege over the Targets.

The Targets list of an Access Control Entry MAY grant a given privilege to more than one Target. An empty Targets list SHALL mean that every Cluster on every Endpoint exposed by the Node is accessible using the granted privilege to any matching Subject. Each Target in the Targets list SHALL specify Cluster instances directly by Cluster ID (on any Endpoint, or limited to particular Endpoints), indirectly by Endpoint ID (all Cluster instances on that Endpoint), or indirectly by Device Type ID (all Cluster instances on all Endpoints containing that Device Type).

For both the Subjects list and Targets list of an Access Control Entry, empty lists permit a rudimentary form of "wildcard" behavior, which is especially useful for codifying policies providing common view/read/discover access to a given subset of Nodes based on Authentication Mode.

CAUTION	Given that "wildcard" (that is, any subject/target) granting is possible with an empty Subjects list or an empty Targets list, it follows that care must be taken by

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Administrators generating and distributing Access Control Lists to ensure unin! tended access does not arise. It is RECOMMENDED to avoid updating Access Control Entries in such a way as to remove Subjects or Targets one by one, which may result in a wildcard after individual actions. Rather, entire Tar! gets/Subjects lists SHOULD be written atomically in a single action, to ensure a complete final state is achieved, with either wildcard or not, and that no acci! dental wildcards arise. Furthermore, such ACL entries with wildcard subject should be deployed with care as they grant the named privilege level to poten! tially many senders, especially when Group Authentication Mode is specified.

6.6.2.3. Subjects do not correspond to users

The Subjects for an Access Control Entry are logical subjects, configured through policy by an Administrator, including possibly a Commissioner during the commissioning phase. A given imple! mentation of administrative logic MAY assign authentication identities to Nodes directly associated with physical end-users (for example, a mobile device of a given end-user). However, since Nodes are logical networking entities, the specific policy of how Node identities are mapped to physical end-users and physical devices is implementation-specific. Therefore, the access granted by a given NodeÕs Access Control system should not be construed as having any particular meaning in regards to physical end-users other than the fact that a given set of Administrators computed a consistent set of Access Control Lists to effect a desired system functionality across all Nodes they administer and end-users they represent.

6.6.2.4. Implementation-specific Rules

Since the target of a given Access Control Entry is a list of Targets, and since Targets (that is, Clusters on Endpoints) are Interaction Model constructs, it should be assumed that access control function! ality as described within this model is constrained to the interaction model layer. However, con! straints on incoming session establishment requests MAY be affected by the Access Control system, based on implementation-defined rules. For example, a Node MAY deny CASE session establishment from an initiator whose identity doesnÕt match any Access Control Entry. These types of rules are implementation-specific and SHOULD be carefully considered, if applied at all. For example, due to the richness of Access Control Entry encoding for Subjects, significant care has to be taken to avoid incorrectly rejecting an incoming CASE session establishment that could be valid. Rejecting valid connections could cause a Node to become unreachable. Any constraints on transport-level and net! work-level functionality, including but not limited to the availability of commissioning-mode con! nectivity, are out of Access Control scope.

6.6.2.5. Incoming Subject Descriptors

The Message Layer SHALL provide sufficient metadata (e.g. Authentication Mode, source identity) about incoming messages to the Interaction Model protocol layer in order to properly enforce Access Control.

An Incoming Subject Descriptor (ISD) is a mapping from the security layer fields of an incoming Message to a tuple of <AuthMode, SubjectDescriptor> that can map unambiguously to an Access Con! trol EntryÕs Subjects and AuthMode fields. See Section 6.6.5.1.2, ÒIncoming Subject Descriptor (ISD) StructureÓ for further details.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

6.6.2.6. Access Control Extensions

An implementation MAY use Access Control Extensions to extend the base Access Control model. Since all extensions are installed by Administrators for a fabric, it is expected that only extensions that would improve overall security will be applied. Since every Vendor MAY implement extensions as they see fit, it SHOULD NOT be expected that an extension will be supported by every Node. It is therefore RECOMMENDED that careful consideration of interoperability concerns be given when implementing Access Control Extensions. A fabricÕs Administrators MAY always read a given NodeÕs Access Control Entries and Access Control Extensions pertaining to the fabric. Therefore, Adminis! trators MAY use extensions to record auditing metadata about Access Control Entries which are not for operational use by the Node.

A Node SHALL preserve every field of the installed Access Control Cluster, including extensions when present, without internally-initiated modifications, so that they may be read-back verbatim upon receiving an appropriate request from an Administrator.

6.6.2.7. Application-level Permissions

The Access Control Cluster SHALL NOT be used to encode application-level permissions and config! urations such as smart lock PIN codes or similar user-facing security functionality. Application-level security is best served by finer-grained capabilities described and addressed by application-domain-specific clusters.

6.6.2.8. Bootstrapping of the Access Control Cluster

Updates to the Access Control List through Access Control Cluster attributes and commands SHALL be restricted by the same Access Control mechanisms as all other clusters on the Node, and there! fore require a grant of Administer privilege. Administrators are able to bootstrap a NodeÕs Access Control List during the commissioning phase due to the Access Control Privilege Granting algorithm implicitly granting the Administer privilege to Administrative Subject Nodes over a PASE commis! sioning channel; this implicit privilege grant applies for the Commissioner to administer the Com! missionee, but not in the opposite direction.

6.6.2.9. Action attribution

The recording of a given Interaction Model ActionÕs attribution to a source entity is distinct from the contents of an Access Control Entry. Action Attribution SHALL be recorded against the Incoming Subject Descriptor (see Section 6.6.5.1.2, ÒIncoming Subject Descriptor (ISD) StructureÓ) rather than against any matched Access Control EntryÕs contents.

6.6.2.10. Restrictions on Administer Level Privilege Grant

Since the Administer privilege level grants wide access to a Node for a given Subject, it SHALL NOT be valid to have an Administer privilege set on an Access Control Entry, unless the AuthMode's Auth! ModeCategory is "CASE". For example, an AuthModeCategory of "Group", which admits no source Node authentication and reduced attribution ability, SHALL NOT be used to grant Administer privilege.

6.6.3. Access Control List Examples

The following Access Control Lists illustrate the flexibility of codifying access control policy using

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

concrete examples.

Upon Factory Data Reset, the Access Control Cluster is empty, having an Access Control List with no entries.

```
Access Control Cluster: {
Ê ACL: [],      // empty
Ê Extension: [] // empty (omitted hereafter)
}
```

However, the Access Control Privilege Granting algorithm behaves as if, over a PASE commissioning channel during the commissioning phase, the following implicit Access Control Entry were present on the Commissionee (but not the Commissioner) to grant Administer privilege for the entire Node.

```
Access Control Cluster: {
Ê ACL: [
Ê   0: {                  // implicit entry only; does not explicitly exist!
Ê     FabricIndex: 0,       // not fabric-specific
Ê     Privilege: Administer,
Ê     AuthMode: PASE,
Ê     Subjects: [],
Ê     Targets: []           // entire node
Ê   }
Ê ]
}
```

During the commissioning phase, the AddNOC command automatically creates an Access Control Entry granting Administer privilege for the entire Node, the appropriate CASE authenticated Subject (in this case, Node ID 0xAAAA_AAAA_AAAA_AAAA) on the appropriate Fabric (in this case, Fabric 0xFAB0_0000_0000_001D as Fabric index 1).

```
Access Control Cluster: {
Ê ACL: [
Ê   0: {
Ê     FabricIndex: 1,
Ê     Privilege: Administer,
Ê     AuthMode: CASE,
Ê     Subjects: [ 0xAAAA_AAAA_AAAA_AAAA ],
Ê     Targets: []
Ê   }
Ê ]
}
```

An Administrator adds an Access Control Entry which grants View privilege, for the entire Node, to all CASE authenticated Nodes.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

```
Access Control Cluster: {
Ê ACL: [
Ê   ...
Ê   1: {
Ê     FabricIndex: 1,
Ê     Privilege: View,
Ê     AuthMode: CASE,
Ê     Subjects: [],
Ê     Targets: []
Ê   }
Ê ]
}
```

An Administrator adds an Access Control Entry which grants Manage privilege, for endpoints 1 and 3, to any Nodes which can authenticate as members of Group 1.

```
Access Control Cluster: {
Ê ACL: [
Ê   ...
Ê   2: {
Ê     FabricIndex: 1,
Ê     Privilege: Manage,
Ê     AuthMode: Group,
Ê     Subjects: [ 0x0000_0000_0000_0001 ],
Ê     Targets: [ { Endpoint: 1 }, { Endpoint: 3 } ]
Ê   }
Ê ]
}
```

An Administrator revises this Access Control Entry to grant the same privilege, for only the pump configuration and control cluster (0x0202) on endpoint 3, and for any door lock cluster (0x0101) on the entire Node, to the same Nodes.

```
Access Control Cluster: {
Ê ACL: [
Ê   ...
Ê   2: {
Ê     FabricIndex: 1,
Ê     Privilege: Manage,
Ê     AuthMode: Group,
Ê     Subjects: [ 0x0000_0000_0000_0001 ],
Ê     Targets: [ { Endpoint: 1 }, { Endpoint: 3, Cluster: 0x0000_0202 }, { Cluster:
0x0000_0101 } ]
Ê   }
Ê ]
}
```

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

An Administrator adds an Access Control Entry which grants Operate privilege, for all endpoints containing the extended color light device (0x010D) on the entire Node, to CASE authenticated Nodes 0x1111_1111_1111_1111 and 0x2222_2222_2222_2222.

```
Access Control Cluster: {
Ê ACL: [
Ê   ...
Ê   3: {
Ê     FabricIndex: 1,
Ê     Privilege: Operate,
Ê     AuthMode: CASE,
Ê     Subjects: [ 0x1111_1111_1111_1111, 0x2222_2222_2222_2222 ],
Ê     Targets: [ { DeviceType: 0x0000_010D } ]
Ê   }
Ê ]
}
```

A Commissioner adds four more Nodes into an existing fabric. These new Nodes have Node IDs 0x3333_3333_3333_3333, 0x4444_4444_4444_4444, 0x5555_5555_5555_5555 and 0x6666_6666_6666_6666 respectively. The Fabric Administration policy requires associating these four nodes and an existing node (0x2222_2222_2222_2222) into a CAT group.

To achieve this, an Administrator will issue NOCs to all five nodes in this CAT group with a CAT value of 0xABCD_0001, which is tag identifier value 0xABCD and version 1, and encoded as subject value 0xFFFF_FFFD_ABCD_0001. To distribute these CATs, an Administrator obtains NOCs from its certificate authority with the requisite subjects including the desired CAT. They are either initially provisioned with the AddNOC command during initial commissioning (for the new Nodes) or updated with UpdateNOC (for existing Nodes).

Then the Administrator grants permissions to the five nodes by updating the ACL of all relevant tar! gets by adding an entry with subject of CAT (0xFFFF_FFFD_ABCD_0001). The Administrator may also remove entries where Node 0x2222_2222_2222_2222 appears as an explicit Subject if presenta! tion of the CAT identifier value 0xABCD and version value 0x0001 confers an equivalent privilege. Note that any Node with CAT identifier value of 0xABCD and version value 0x0001 or higher in their NOC will have this privilege.

```
Access Control Cluster: {
Ê ACL: [
Ê   ...
Ê   3: {
Ê     FabricIndex: 1,
Ê     Privilege: Operate,
Ê     AuthMode: CASE,
Ê     Subjects: [ 0x1111_1111_1111_1111, 0xFFFF_FFFD_ABCD_0001],
Ê     Targets: [ { DeviceType: 0x0000_010D } ]
Ê   }
Ê ]
}
```


Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

An Administrator wants to remove the Node with Node ID 0x3333_3333_3333_3333 from the CAT group defined by CAT identifier value of 0xABCD, as installed in the previous example.

The Administrator could follow the steps outlined below:

1. Administrator will ensure that the removed node having Node ID 0x3333_3333_3333_3333 will not receive a new NOC with an CAT identifier value of 0xABCD. Note that the node removed from the group will continue to hold existing NOC (with CAT identifier of 0xABCD and version 0x0001).
2. An Administrator updates NOCs with CAT identifier value of 0xABCD and version 0x0002 (encoded as subject 0xFFFF_FFFD_ABCD_0002) in all remaining currently reachable Nodes within the CAT group to ensure they continue to have same privilege as before. In this example, Nodes having Node IDs 0x2222_2222_2222_2222, 0x4444_4444_4444_4444 and 0x5555_5555_5555_5555 are currently reachable.
3. Node with Node ID 0x6666_6666_6666_6666 from this CAT group is a valid member but was not reachable by an Administrator at the time of this change. This Node will continue to hold exist! ing NOC with CAT of 0xABCD_0001.
4. After updating NOCs of all reachable Nodes, the Administrator SHOULD revise the Access Con! trol Entry of all reachable nodes who have the previous CAT (encoded as subject 0xFFF! F_FFFD_ABCD_0001) in an ACL entry, to remove privilege from the Node no longer in the group! ing (i.e. those with version 0x0001) by increasing trusted version value to be higher than 0x0001. The Administrator decides to increment version value by one to set the new version value to be 0x0002.
5. Once ACL changes are propagated to all controlled nodes, they will no longer allow access privi! leges to any Node with older version (i.e. value less than 0x0002) of CAT identifier value 0xABCD. Hence, the node removed from the group, having Node ID 0x3333_3333_3333_3333 and CAT with identifier 0xABCD and version of 0x0001, can no longer access any of the controlled nodes whose ACL entries were updated to have a subject of 0xFFFF_FFFD_ABCD_0002 (CAT identifier value 0xABCD, version 0x0002).
6. Node having Node ID of 0x6666_6666_6666_6666 will not be able to access any Nodes by relying on CAT, since it does not have an NOC with latest CAT (with version 0x0002). However, it can still access Nodes that list it as a subject Node ID explicitly. When an Administrator eventually estab! lishes connection to this Node, the Administrator SHOULD update the NOC to the latest version, with CAT set to 0xABCD_0002. After having its NOC updated to have the newest version of the CAT, the Node with Node ID 0x6666_6666_6666_6666 will again have access to Nodes that list subject 0xFFFF_FFFD_ABCD_0002 (CAT identifier value 0xABCD, version 0x0002), with no fur! ther updates to ACL entries of existing Nodes.
7. Any controlled Node which previously held an ACL Entry with prior version of the updated CAT (subject 0xFFFF_FFFD_ABCD_0001) but was not reachable by an Administrator at the time of update, will continue to hold the previous Access Control Entry with a subject allowing CAT with identifier of 0xABCD and version 0x0001 or higher. Thus, these Nodes will grant privileges to any Node from the original CAT group (including Node ID 0x3333_3333_3333_3333). When an Administrator eventually establishes connection to this Node with older ACL entry, the Adminis! trator SHOULD update it with the latest value, so that Node ID 0x3333_3333_3333_3333 no longer has privileges.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Note that in the above example, the CAT identifier value remained the same (0xABCD) in NOCs and ACL entries throughout these steps. Only the version portion was updated to effect changes to the meaning of the CAT.

As can be seen in the example above, there are multiple steps involving updates to NOCs and ACL entries to affect CAT-based grouping and aliasing policies. It is therefore possible that some Nodes may not receive these changes immediately, due to network reachability issues, such as being pow! ered down for an extended period, and thus have ACL entries or NOCs that grant temporarily obso! lete privileges. This is true as well with direct Node ID subjects, in general.

Administrators SHOULD aim for best-effort eventual consistency while executing the steps outlined above.

```
Access Control Cluster: {
Ê ACL: [
Ê   ...
Ê   3: {
Ê     FabricIndex: 1,
Ê     Privilege: Operate,
Ê     AuthMode: CASE,
Ê     Subjects: [ 0x1111_1111_1111_1111, 0xFFFF_FFFD_ABCD_0002],
Ê     Targets: [ { DeviceType: 0x0000_010D } ]
Ê   }
Ê ]
}
```

6.6.4. Access Control Cluster update side-effects

Updates to the Access Control Cluster SHALL take immediate effect in the Access Control system. For example, given an Interaction Model action message containing the following actions, the Access Control Privilege Granting algorithm would grant a privilege of None for the second action, since the first action would take effect immediately beforehand.

¥ Pre-conditions:

¥ Access Control List has single entry: [{Privilege: Administer, Authmode: CASE (2), Subjects: [0x0011223344556677], Targets: []}]

¥ Node ID 0x0011223344556677 over CASE is allowed Administer privilege for all targets

¥ Incoming message Source is Node ID 0x0011223344556677 over CASE: matches Access Control Entry subject

¥ Actions:

1. Action: Write (1st path)
    1. Path: Endpoint[0]/Cluster[AccessControl]/Attribute[ACL]/ListIndex[0]/Field[Targets]
    2. Value: [{Endpoint: 2}]

        & Single entry updated to target only Endpoint 2
    3. Granted : Administer privilege granted, due to Access Control Entry match

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

2. Action: Write (2nd path)
    1. Path: Endpoint[1]/Cluster[OnOff]/Attribute[OnTime]
    2. Denied : No privilege granted, because prior action in the same message had updated Access Control List to only allow access to Endpoint 2, and this action targets Endpoint 1

¥ Post-conditions:

¥ Access Control List has single entry, updated by first path of Write Action: [{Privilege: Adminis! ter, Authmode: CASE (2), Subjects: [0x0011223344556677], Targets: [{Endpoint: 2}]}]

¥ Node ID 0x0011223344556677 over CASE is allowed Administer privilege for only Endpoint 2 tar! get

Note that in this example, the Node has inadvertently lost its ability to update the Access Control Cluster by limiting its Administer privilege to Endpoint 2.

6.6.5. Conceptual Access Control Privilege Granting Algorithm

This section describes an overall Conceptual Access Control Privilege Granting algorithm. Imple! mentations of this algorithm SHALL have an identical outcome to the output of this conceptual algorithm described below.

The Interaction Model protocol, through its message handling, SHALL determine the privilege level granted per Target, on every instance where a Target is referenced for use.

6.6.5.1. Necessary Data Structures

6.6.5.1.1. Access Control List

The Access Control List contains several Access Control Entries , previously described in Section 6.6.2, ÒModelÓ.

The entry fields are:

¥ Subjects List (SubjectID [] Subjects)

¥ Targets List (TargetStruct[] Targets)

¥ Authentication Mode value (AuthModeEnum AuthMode)

¥ Privilege value (PrivilegeEnum Privilege)

6.6.5.1.2. Incoming Subject Descriptor (ISD) Structure

Each incoming message has a unique <AuthMode, SubjectDescriptor> applicable to it, whose deriva! tion is deterministic based on both incoming message fields and session metadata fields. For exam! ple, if a message arrives that matches a given CASE Session ID, then the metadata for that CASE ses! sion would be used.

Computation of the ISD is described in Section 6.6.5.3, ÒDerivation of ISD from Incoming MessageÓ .

The ISD fields are as follows:

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

¥ Commissioning Flag (bool IsCommissioning), whether the authentication is over a commission! ing channel.

¥ Authentication Mode (AuthModeEnum AuthMode), mapping to an authentication mode, directly com! parable to Access Control Entry AuthMode .

¥ Subjects List (list<SubjectID> Subjects), mapping incoming message source to a type of subject, such as a CASE session Source Node ID.

¥ Fabric Index (FabricIndex FabricIndex), mapping to a fabric reference.

6.6.5.2. Overall Algorithm

The algorithm takes as input:

¥ the ISD of Incoming Message (subject_desc)

¥ the Endpoint ID (endpoint_id) for which the querier requires a Privilege level

¥ the Cluster ID (cluster_id) for which the querier requires a Privilege level

¥ the Access Control List (acl) from the Access Control Cluster

The output of the algorithm is:

¥ A set of privileges granted for the Action Path, which is a subset of {View, ProxyView, Operate, Manage, Administer} as described in AccessControlEntryPrivilegeEnum .

The computation of the ISD is a pre-condition to the algorithm and is described in Section 6.6.5.3, ÒDerivation of ISD from Incoming MessageÓ .

The goal is to find the complete set of privileges granted given the input. The principle of least privi! lege is respected by virtue of the entire Access Control List having been computed with rules such that the least privilege is granted to all subjects. Therefore, any Access Control Entry granting the required privilege to the subject for a given target is sufficient to determine whether access is allowed.

The algorithm SHALL function as follows:

```
def subject_matches(acl_subject, isd_subject):
Ê # Subjects must match exactly, or both are CAT
Ê # with matching CAT ID and acceptable CAT version
Ê return (acl_subject == isd_subject) or
Ê   (is_cat(acl_subject) and is_cat(isd_subject) and
Ê    (get_cat_id(acl_subject) == get_cat_id(isd_subject)) and
Ê    (get_cat_version(isd_subject) >= get_cat_version(acl_subject))

def add_granted_privilege(granted_privileges, privilege):
Ê # Add the new privilege to the granted privileges set
Ê granted_privileges.add(privilege)
Ê # Also add any privileges subsumed by the new privilege
Ê if (privilege == PrivilegeEnum.ProxyView):
Ê   granted_privileges.add(PrivilegeEnum.View)
Ê elif (privilege == PrivilegeEnum.Operate):
```

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

```
Ê   granted_privileges.add(PrivilegeEnum.View)
Ê elif (privilege == PrivilegeEnum.Manage):
Ê   granted_privileges.add(PrivilegeEnum.Operate)
Ê   granted_privileges.add(PrivilegeEnum.View)
Ê elif (privilege == PrivilegeEnum.Administer):
Ê   granted_privileges.add(PrivilegeEnum.Manage)
Ê   granted_privileges.add(PrivilegeEnum.Operate)
Ê   granted_privileges.add(PrivilegeEnum.ProxyView)
Ê   granted_privileges.add(PrivilegeEnum.View)

def get_granted_privileges(acl, subject_desc, endpoint_id, cluster_id) ->
set[Privilege]:
Ê # Granted privileges set is initially empty
Ê granted_privileges = set()

Ê # PASE commissioning channel implicitly grants administer privilege to commissioner
Ê if subject_desc.AuthMode == AuthModeEnum.PASE and
subject_desc.IsCommissioneeDuringCommissioning:
Ê   add_granted_privilege(granted_privileges, PrivilegeEnum.Administer)

Ê for acl_entry in acl:
Ê   # End checking if highest privilege is granted
Ê   if PrivilegeEnum.Administer in granted_privileges: break

Ê   # Fabric index must match, there are no valid entries with FabricIndex == 0
Ê   # other than the implicit PASE entry, which we will not see explicitly in the
Ê   # access control list
Ê   if acl_entry.FabricIndex == 0: continue
Ê   if acl_entry.FabricIndex != subject_desc.FabricIndex: continue

Ê   # Auth mode must match
Ê   if acl_entry.AuthMode != subject_desc.AuthMode: continue

Ê   # Subject must match, or be "wildcard"
Ê   if is_empty(acl_entry.Subjects):
Ê     # Precondition: only CASE and Group auth can have empty subjects
Ê     assert(acl_entry.AuthMode in [AuthModeEnum.CASE, AuthModeEnum.Group])
Ê     # Empty is wildcard, no match required
Ê   else:
Ê     # Non-empty requires a match
Ê     matched_subject = False
Ê     for acl_subject in acl_entry.Subjects:
Ê       for isd_subject in subject_desc.Subjects:
Ê         if subject_matches(acl_subject, isd_subject):
Ê           matched_subject = True
Ê           break
Ê       if matched_subject: break
Ê     if not matched_subject: continue

Ê   # Target must match, or be "wildcard"
Ê   if is_empty(acl_entry.Targets):
```


Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

```
Ê    # Empty is wildcard, no match required
Ê   else:
Ê    # Non-empty requires a match
Ê    matched_target = False
Ê    for target in acl_entry.Targets:
Ê     # Precondition: target cannot be empty
Ê     assert(target.Cluster != null or target.Endpoint != null or target.DeviceType
!= null)
Ê     # Precondition: target cannot specify both endpoint and device type
Ê     assert(target.Endpoint == null or target.DeviceType == null)
Ê     # Cluster must match, or be wildcard
Ê     if target.Cluster != null and target.Cluster != cluster_id:
Ê      continue
Ê     # Endpoint must match, or be wildcard
Ê     if target.Endpoint != null and target.Endpoint != endpoint_id:
Ê      continue
Ê     # Endpoint may be specified indirectly via device type
Ê     if target.DeviceType != null and not
endpoint_contains_device_type(endpoint_id, target.DeviceType):
Ê      continue
Ê     matched_target = True
Ê     break
Ê    if not matched_target: continue

Ê   # Extensions processing must not fail
Ê   if not extensions_are_valid(acl, acl_entry, subject_desc, endpoint_id,
cluster_id): continue

Ê   # All checks have passed, add privilege to granted privilege set
Ê   add_granted_privilege(granted_privileges, acl_entry.privilege)

Ê # Should never grant Administer privilege to a Group.
Ê if subject_desc.AuthMode == AuthModeEnum.Group:
Ê   assert (PrivilegeEnum.Administer not in granted_privileges)

Ê return granted_privileges
```

6.6.5.3. Derivation of ISD from Incoming Message

The algorithm to derive the ISD from an incoming message takes as input:

- ¥ The incoming message (message)
- ¥ The Session ID of the incoming message (session_id)
- ¥ A conceptual Sessions Metadata database (sessions_metadata)
- ¥ The Group Key Management Cluster (group_key_management_cluster)

The output of the algorithm is the SubjectDescriptor structure below:

```
DEFAULT_COMMISSIONING_PASSCODE = 0
```

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

```
enum AuthModeEnum {
Ê None = 0, # conceptual "no auth" value
Ê PASE = 1,
Ê CASE = 2,
Ê Group = 3
}

struct SubjectDescriptor {
Ê bool IsCommissioning;
Ê AuthModeEnum AuthMode;
Ê list<SubjectID> Subjects; # max 3 items
Ê FabricIndex FabricIndex;
}
```

The algorithm SHALL function as follows:

```
def get_isd_from_message(message) -> SubjectDescriptor:
Ê isd = {
Ê   IsCommissioning: False,
Ê   AuthMode: AuthModeEnum.None,
Ê   Subjects: [],
Ê   FabricIndex: 0
Ê }

Ê session_id = message.get_session_id()

Ê if sessions_metadata.get_auth_mode(session_id) == AuthModeEnum.PASE:
Ê   isd.AuthMode = AuthModeEnum.PASE
Ê   isd.IsCommissioning = True
Ê   isd.Subjects.append(DEFAULT_COMMISSIONING_PASSCODE)
Ê   isd.FabricIndex = sessions_metadata.get_fabric_index(session_id) # may be zero
Ê else if sessions_metadata.get_auth_mode(session_id) == AuthModeEnum.CASE:
Ê   isd.AuthMode = AuthModeEnum.CASE
Ê   isd.Subjects.append(sessions_metadata.get_src_node_id(session_id))
Ê   # CASE session may contain CATs which also serve as subjects
Ê   # Append all CATs if present (can be up to 3)
Ê   if sessions_metadata.has_src_case_authenticated_tags(session_id):
Ê
isd.Subjects.append(sessions_metadata.get_src_case_authenticated_tags(session_id))
Ê   isd.FabricIndex = sessions_metadata.get_fabric_index(session_id)
Ê   assert(isd.FabricIndex != 0) # cannot be zero
Ê else if sessions_metadata.get_auth_mode(session_id) == AuthModeEnum.Group:
Ê   # Message is assumed to have been decrypted and matched properly prior to
Ê   # this procedure occurring.
Ê   group_id = message.get_dst_group_id()
Ê   group_key_id = sessions_metadata.get_group_key_id(message)
Ê   # Group membership must be verified against Group Key Management Cluster
Ê   if group_key_management_cluster.group_key_map_has_mapping(group_id, group_key_id):
Ê     isd.AuthMode = AuthModeEnum.Group
```


Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

```
Ê    isd.Subjects.append(group_id)
Ê    isd.FabricIndex = sessions_metadata.get_fabric_index(message)
Ê    assert(isd.FabricIndex != 0) # cannot be zero
Ê else:
Ê  # Do nothing on error, ISD remains unchanged
Ê  assert(isd.IsCommissioning == False)
Ê  assert(isd.AuthMode == AuthModeEnum.None)
Ê  assert(is_empty(isd.Subjects))
Ê  assert(isd.FabricIndex == 0)

Ê return isd
```

6.6.6. Applying Privileges to Action Paths

The Data Model specifies which privilege is required for each data element, via its access qualities.

The Interaction Model specifies how each action is processed, for both its request and its response. This includes details on how the Interaction Model uses Access Control to determine whether to allow the request (i.e. continue processing), or to deny the request (and whether/how that is indi! cated in the response).

Determining whether to allow or deny an action for a request path entails:

- ¥ Determining the required privilege for the action, given the request path and type of access requested;
- ¥ Determining the set of granted privileges for the action, given the request path and requesting subject;
- ¥ Checking whether the required privilege is present in the set of granted privileges:
  - %If present, the action is allowed;
  - %If not present, the action is denied.

Note that the Interaction Model may allow the action for some request paths while denying it for other request paths in the same action. Also, note that Access Control is merely one of the checks used by the Interaction Model, and an action that is allowed by Access Control may fail for other reasons.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Chapter 7. Data Model Specification

7.1. Practical Information

7.1.1. Revision History

Revision	Description
1-15	Released as revisions of the Dotdot Architecture Model specification
16	Initial Release of this specification

7.1.2. Scope & Purpose

This is part of a package of Data Model specifications that are agnostic to underlying layers (encod! ing, message, network, transport, etc.). Each specification below may be independently maintained. This package, as a whole, shall be independently maintained as agnostic and decoupled from lower layers. This package may be referenced by inclusion in vertical protocol stack specifications.

Data Model	Defines first order elements and namespace for endpoints, clusters, attributes, data types, etc.
Interaction Model	Defines interactions, transactions and actions between nodes.
System Model	Defines relationships that are managed between endpoints and clusters.
Cluster Library	Reference library of cluster specifications.
Device Library	Reference library of device type specifications.

7.1.3. Origin Story

The origin of this section is the Dotdot Architecture Model [Dotdot Architecture] and parts of Chap! ter 2 of the Zigbee Cluster Library specification [ZCL] that define the data model.

The purpose of this document is to extend and better define the data model architecture, while not breaking the certifiable cluster specifications in the Zigbee Cluster Library (currently at revision 8). Under the Matter project, new and existing clusters and device types may take advantage of extended architecture elements. Ultimately, the plan is that this architecture is available and main! tained for all underlying certifiable protocol stacks.

7.1.4. Overview

This document defines first order elements and namespace of the data model and can also be called the meta-model (of the data model). This document is the "read me first" specification in the data model. This data model is ultimately implemented in the application layer of a communication stack.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

This specification does not define how data is stored, encoded or what interactions are allowed on the data.

7.1.5. Glossary

Term	Description
MS	Manufacturer or Vendor Specific
N/A	not applicable
desc	see detailed description section

7.1.6. Conventions

The following conventions are used in Data Model specifications.

7.1.6.1. Enumerations and Reserved Values

An undefined value or range of an enumeration, field, or identifier SHALL be considered reserved for future revisions of this standard and SHALL NOT be available for implementation. It is RECOM! MENDED that a value stay undefined, rather than defining it as "reserved".

A value or range of an enumeration, field, or identifier that is available for non-standard imple! mentation SHALL be defined as manufacturer specific.

A value or range of an enumeration, field, or identifier that is available for other parts of this stan! dard SHALL be defined as such.

A value or range of an enumeration, field, or identifier that is deprecated, and not available for implementation, SHALL be defined as deprecated.

7.1.7. Reserved Bit Fields

- ¥ An undefined bit or bit field SHALL be considered reserved for future revisions of this standard and SHALL not be available for implementation.
- ¥ It is RECOMMENDED that a bit stay undefined, rather than defining it as "reserved".
- ¥ An implementation of a revision where a bit is reserved SHALL indicate that bit as zero when conveying that bit in an interaction, and ignore that bit when conveyed from another imple! mentation.

7.1.7.1. Number Format

In this specification, hexadecimal numbers are prefixed with the designation Ò0xÓ and binary num! bers are prefixed with the designation Ò0bÓ. All other numbers are assumed to be decimal unless indicated otherwise within the associated text.

Binary numbers are specified as successive groups of 4 bits, separated by a space (Ò Ò) character from the most significant bit (next to the 0b prefix and leftmost on the page) to the least significant bit (rightmost on the page), e.g. the binary number 0b0000 1111 represents the decimal number 15.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Where individual bits are indicated (e.g. bit 3) the bit numbers are relative to the least significant bit which is bit 0.

When a digit is specified as having any value in the range of that digit, it is specified with an ÒxÓ (this should not be confused with the "x" in the prefix "0x" for hexadecimal notation).

For example:

- ¥ Ò0b0000 0xxxÓ indicates that the lower 3 bits can take any value but the upper 5 bits must each be set to 0.
- ¥ Ò0x0000 0xxxÓ indicates that the lower 3 nibbles can take any value but the upper 5 nib! bles must each be set to 0.

7.2. Data Qualities

Cluster specifications and device type definitions have tables that define the qualities of elements that make up the cluster or device type. Not all elements have all qualities. For example, a com! mand does not have the read quality. Some elements have intrinsic qualities, that are not listed. For example, an event always has the read quality. Qualities SHALL be defined in columns in tables that describe data model elements.

7.2.1. Common Data Table Columns

The following columns are common across tables describing attributes, commands, events and structs:

ID

Defines an identifier for the data model element that is unique at its context.

Name

Defines a CamelCase name of the element to be used in specification text, not the protocol. Text usage SHALL always be followed with the element name (e.g. CurrentLevel attribute, Stopped event, or Left field).

Field

Same as Name . Other headers like "Field", "Bit Field", and "Command Field", are deprecated. Use Name .

Conformance

Defines dependencies on whether an element is optional or mandatory.

Access

Defines how an element is accessed (e.g. read or write) and what privileges are required to access the data.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

7.2.2. Other Data Table Columns

Other columns specific to the element:

Data Type

A data field requires this column for attribute, event or command data.

Other Qualities

This is a catchall column for uncategorized qualities.

Default

This defines a default value for data fields.

Response

Cluster command tables have this column.

Direction

Cluster command tables have this column.

Priority

Event tables have this column.

Value

Enumerations use this column instead of the ID column.

7.3. Conformance

A Conformance column defines optionality and dependency for any data model element or set of elements. This column is valid for attributes, commands, events, enumerations, and fields of com! mands, events or structures.

Conformance Column	Name	Description
M	Mandatory	This is part of the base manda! tory feature set and is manda! tory for current revision.
O	Optional	This is a purely optional ele! ment with no dependencies, except the M set.
P	Provisional	See Provisional below.
D	Deprecated	This is a deprecated item that MAY occur in legacy implemen! tations, but not in the current revision.
X	Disallowed	This is disallowed for the clus! ter derivation.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Conformance Column	Name	Description
AB	Mandatory	This SHALL be supported if AB is true. AB is a boolean expres! sion
[AB]	Optional	This MAY be supported only if AB is true. Brackets also act as parentheses.
EF	Operand	True if EF feature or element is supported.
EF==v	Equal	True if EF is equal to the fixed and non-changing value v.
EF!=v	UnEqual	True if EF does not equal the non-changing value v.
AB | CD	Or	True if either AB or CD is true.
AB ^ CD	Xor	True if only one of AB or CD is true, not both.
AB & CD	And	True if both AB and CD are true.
!AB	Exclusivity	True if AB is false.
(AB & CD)	Parentheses	Parentheses can be put around any conformance expression to combine.
C1, C2É	List	A conformance that is a list of boolean expressions.
C.an	Choice	Exclusive choices between a number of elements with the same conformance

7.3.1. Optional

Optionality with "[]" SHALL be defined for an entire expression and not for parts of an expression. Individual conformance list entries MAY define optionality with "[]", but not the entire list.

For example: The expression "[AA] & BB" is illegal, however, "[AA & BB]" or "AA & BB" is legal.

For example: The expression "AA | [BB]" is illegal, however, "[AA | BB]" or "AA, [BB]" is legal.

The tag "O" SHALL define the element as optional for the revision. "O" SHALL only be used by itself, without operators, to mean optional without dependencies, or SHALL be used in a conformance list ending in ", O", to mean otherwise optional.

7.3.2. Provisional

The tag "P" defines the element as provisional. "P" may be used in a list, where the intended confor! mance or list follows the "P". If the intended conformance has not been determined, then nothing

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

appears after the "P".

It is recommended that the intended future conformance be noted, so that when the provisional marking is removed, the intended conformance becomes the current conformance.

For example: "P, M" means provisional, but mandatory, when not provisional in the future.

For example: "P, [AA & BB]" means provisional for now, but optional if AA & BB are true, when not provisional in the future.

For example: "[AA], P" means optional for AA and provisional otherwise, where the future confor! mance is unknown at this time.

Each provisional element shall be listed in a higher level specification, that includes the data model, and data model derived specifications that use this notation.

7.3.3. Mandatory

The tag "M" SHALL define the element as mandatory. "M" SHALL only be used by itself, without operators, to mean mandatory without dependencies, or SHALL be used in a conformance list end! ing in ", M", to mean otherwise mandatory.

7.3.4. Disallowed

The tag "X" is used when a derived cluster removes support for some elements. The tag "X" SHALL define the element as disallowed for the revision of the derivation. "X" SHALL only be used by itself, without operators.

7.3.5. Deprecated

The tag "D" SHALL define the element as disallowed for the revision. Previous revisions MAY sup! port this element and conformance is defined in such previous revisions. "D" SHALL only be used by itself, without operators.

7.3.6. Exclusivity

Exclusivity occurs when the entire expression only excludes.

It is recommended to not use exclusive conformance. Positive conformance is recommended.

For example: Excluding an element with "!Matter" means that it is mandatory otherwise. Better to use positive conformance with (e.g. "Zigbee"). For example: Excluding an element with "[!Matter]" means that it is optional otherwise. Better to use positive conformance with (e.g. "[Zigbee]").

7.3.7. List

A conformance list is evaluated from top to bottom, and left to right, where an expression is mutu! ally exclusive to the previous expressions (above and to the left). It is a shorthand that allows defin! ing conformance which depends on the previously evaluated true expression.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Some examples:

¥ "AB, O" means mandatory for AB and optional otherwise.

¥ "AB, [CD]" means mandatory for AB, optional if CD is true and AB is false, otherwise not allowed.

¥ "!AB, O" means mandatory if AB is false, otherwise optional (if AB is true).

¥ "[AB], M" is the equivalent to "!AB, O", and a clearer way to define the conformance.

¥ "[AA], [BB], [CC]" is the equivalent to "[AA | BB | CC]".

If a row in a table is repeated for an element with qualities that change based on tags, then evalua! tion top to bottom is also supported. For the example below, the MinLevel element is mandatory for LT with the minimum of 1 (not zero), mandatory for AB with the minimum of 0 (zero), otherwise the element is optional with a minimum of 0 (zero).

ID	Name	Type	Constraint	Default	Conformance
43	MinLevel	uint8	1 to 100 0 to 100	1 0	LT AB, O

Above is logically the same and a shorthand for using the choice conformance notation below.

ID	Name	Type	Constraint	Default	Conformance
43	MinLevel	uint8	1 to 100	1	LT.a
43	MinLevel	uint8	0 to 100	0	(AB, O).a

7.3.8. Expressions and Optionality

A conformance expression supports conformance tags as operands. A conformance tag for a cluster MAY be the name of a cluster feature (see FeatureMap Attribute). A conformance tag for a cluster MAY be the name of an element in the Name column of the same table. A conformance tag for a device type definition MAY also include a condition of the node.

Expressions SHALL represent a dependency with boolean logic using:

¥ the NOT operator such as "!AA"

¥ the OR operator such as ÒAA | BBÓ

¥ the AND operator such as ÒAA & BBÓ

¥ the XOR (exclusive or) operator such as "AA ^ BB"

¥ the equal operator such as "AA==10"

¥ the not equal operator such as "AA!=10"

Equality operators require that a value can be resolved for the left operand. If the left operand is not supported, the default is the value. "null" is a valid value for the equality operator.

Simple dependencies MAY also be defined in conformance. If a Max attribute has a dependency on a Min attribute, then the conformance for Max is "Min". Exclusive logic also applies. For example, if the Absolute attribute is mutually exclusive to the Percentage attribute, and one of the two must be

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

supported, then conformance for Absolute would be "!Percentage", and conformance for Percentage would be "!Absolute".

Unless overridden with parentheses, the order of operations is:

- ¥ NOT operator "!"
- ¥ AND operator "&", OR operator "|", XOR operator "^"
- ¥ equal "==", not equal "!="

If an expression is false, there SHALL be no assumption of general optionality. If the conformance expression evaluates to false but optional, the expression is false.

Some examples of conformance definitions:

- ¥ "[AB]" means the element is not allowed, if AB evaluates to false.
- ¥ "[AA & BB]" means optional if AA and BB are both true, but excluded (and not optional) otherwise.
- ¥ "[!AA & BB]" means only optional if AA is false and BB is true.

7.3.9. Choice

Choice exclusivity supports conformance when defining support for only one or some elements at the same level. This set of elements is called the choice set.

C is any logical AB expression or conformance list, including "O" (optional) or "M" (mandatory), but not "X" or "D". If C is a conformance list, then the list SHALL be surrounded by parentheses.

a is a lower case letter identifying 2 or more elements in the choice set, that SHALL be at the same scope (in the same table).

n determines the number in the choice set that SHALL be supported after evaluation of the conformance C. n is a number between one (1) and the number of elements tagged with an a, minus one. n may be suffixed with a plus sign ("+"), which means the number or more SHALL be supported, otherwise n is exactly the number that SHALL be supported. If n is omitted, then n is considered to be one.

When this conformance notation is used, each element identifying the set or choices with a, SHALL have a duplicate n value and duplicate the "+" if used.

Invalid example: It is illegal to use "M.a" for one element and "M.a2" for another.

Valid examples:

- ¥ "AB.a" means only one of the "a" elements SHALL be supported.
- ¥ "AB.a2" means exactly two of the "a" elements SHALL be supported.
- ¥ "AB.a2+" means at least two of the "a" elements SHALL be supported.
- ¥ "AB.a+" means at least one of the "a" elements SHALL be supported.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Optionality such as "O.a" SHALL support the choice of not implementing the element, regardless of the n value. However, if the element set conformance evaluates to true, the n value SHALL be sup! ported.

For example: The following is valid, because the n value is always supported.

ID	Name	Type	Constraint	Default	Conformance
43	AbsoluteValue	uint16	0 to 5000	0	AB.a+
44	PercentValue	uint8	0 to 100	0	(CD, O).a+

Different expressions for C are allowed, which may limit the choices, based on conformance, to greater than zero, but less than n. In such cases, consideration is needed to define choice and con! formance so that the n value is satisfied.

For example: The following is invalid if conformance allows AB to be false and CD to be true, because the n value of 2 is not satisfied.

ID	Name	Type	Constraint	Default	Conformance
43	AbsoluteCm	uint16	0 to 5000	0	(AB & CM).a2
44	AbsoluteIn	uint16	0 to 2000	0	(AB & IN).a2
45	Percent	uint8	0 to 100	0	CD.a2

7.3.10. Blank Conformance

If an element does not have a designated conformance (the column is blank or omitted), then it SHALL inherit conformance from the next highest element in the model hierarchy. For example: A data field in a struct attribute inherits its conformance from the attribute.

7.4. Element

An element of the data model is a data construct that supports an instance of data. Listed below are the elements of the data model.

First order elements

fabric, node, endpoint, cluster

Cluster first order elements

command, event, attribute

Nested elements

command field, event field, struct field

Dynamic element

list entry

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Semantic elements

device type, data type

Attribute data

elements (above) that are part of an attribute

Data field

attribute, field element, or list entry (see Data Field)

7.4.1. Encoded Element Processing

When parsing or processing encoded payloads of elements as represented by an encoding layer, such as TLV format , the following general rules apply:

- ¥ Unknown elements SHALL be ignored and skipped. This provides forward compatibility with future elements.
- ¥ Elements SHALL be present when conveyed according to the elementÕs conformance.
- ¥ Elements that are present and conformant SHALL be processed.

7.5. Fabric

A fabric is set of nodes that interact by accessing data model elements as defined in the Interaction Model . A fabric is a security domain that allows a set of nodes to be identified and communicate within the context of the domain. A node is considered to be 'on' a fabric, when it can be identified and interact in the context of that fabric. An interaction is considered to occur 'on' a fabric, when the interaction occurs in the context of that fabric (see Accessing Fabric). Each interaction occurs either on a single fabric, or without a fabric context (see Accessing Fabric).

A node MAY be identified and interact on one or more fabrics.

How a fabric is established and how a node comes to be on a fabric is not defined here and left to the lower layers.

7.5.1. Accessing Fabric

If an interaction is associated with a particular fabric, that fabric is called the "accessing fabric".

If the interaction is not associated with a fabric, the accessing fabric does not exist. In this case any comparison of the accessing fabric to any existing fabric SHALL consider them not equal.

7.5.2. Fabric-Index

Each fabric supported on a node is referenced by fabric-index that is unique on the node. This fab! ric-index enables the look-up of the full fabric information from the fabric-index. A fabric-index of 0 (zero) or null SHALL indicate that there is no fabric associated with the context in which the fab! ric-index is being used. If fabric-index is used in a context that is exclusively associated with a fab! ric, such as fabric-scoped data model elements, then the fabric-index values SHALL NOT include 0 (zero) or null.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

The fabric-index corresponding to the accessing fabric is called the "accessing fabric-index". If the accessing fabric does not exist, the accessing fabric-index SHALL indicate no fabric with a fabric-index of 0.

7.5.3. Fabric-Scoped Data

Most cluster data instances are accessible regardless of the accessing fabric. However, data that is exclusively associated with a particular fabric SHALL be defined as being fabric-scoped. Fabric-scoped data SHALL be defined with the fabric-scoped quality .

The fabric associated with fabric-scoped data is called the "associated fabric".

Fabric-scoped data allows multiple accessing fabrics to manipulate a list of data items without interfering with each other. See Fabric Filtered List .

Fabric-scoped data SHALL be limited to the following:

- ¥ list of fabric-scoped structs
- ¥ fabric-sensitive event

A fabric-scoped data instance is always a composite struct-like data instance, with multiple fields.

Fabric-scoped data SHALL always include the FabricIndex field to indicate the associated fabric. The FabricIndex field for fabric-scoped data SHALL NOT be 0 or null.

Any interaction, including cluster commands, SHALL NOT cause modification of fabric-scoped data, directly or indirectly, if the interaction has an accessing fabric different than the associated fabric for the data, except in the case of a cluster command that explicitly defines an exception to this rule.

Data that is fabric-scoped may also be fabric-sensitive , all or in part.

7.5.4. Fabric-Scoped IDs

Some data types are fabric-scoped IDs, including, but not limited to, node ID and group ID.

A fabric-scoped ID MAY require the presence of a fabric-index data type field within the same nesting scope to indicate the fabric associated with the ID in these cases:

- ¥ If the fabric-scoped ID is not part of fabric-scoped data.
- ¥ If the fabric-scoped ID is part of fabric-scoped data with an associated fabric that is not the fabric associated with the ID.

Fabric-scoped IDs SHALL only be indicated in these elements:

- ¥ structs
- ¥ events
- ¥ commands

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Where necessary, specification text SHOULD define the data to which the fabric-index applies.

7.6. Access

Data model elements have access qualities. Some elements have intrinsic access or access limita! tions. For example: Cluster commands or command fields are not writable.

An Access column defines access to a data model element or set of elements. This column is valid for attributes, commands, events, and nested attribute data fields.

Access Column	Description
R	Read Access
W	Write Access
R[W]	Read Access and optionally Write Access
R*W	Deprecated: use R[W]
Fabric - separate with space from RW	
F	Fabric-Scoped Quality
S	Fabric-Sensitive Quality
Privileges - separate with space from RW FS	
V	Read Access or Invoke Access requires View privilege
O	Read Access, Write Access , or Invoke Access requires Operate privilege
M	Read Access, Write Access , or Invoke Access requires Manage privilege
A	Read Access, Write Access , or Invoke Access requires Administer privilege
Timed - separate with space from RW FS VOMA	
T	Write Access or Invoke Access with timed inter! action only

Attributes, commands, and events SHALL define their access, and SHALL include privileges in their access definition. For example: An attribute defines whether it is readable or writable, and what privileges are required to do so.

Attributes, commands, and events that do not define any privileges as access qualities SHALL be deemed to have the following:

¥ View privilege required for Read access,

¥ Operate privilege required for Write access,

¥ Operate privilege required for Invoke access for request commands.

¥ No privileges defined for response commands.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

For example: An event with implicit read access or explicit 'R' access defaults to access 'R V'. An attribute with access 'RW' defaults to access 'RW VO'. A request command with implicit invoke access defaults to privilege 'O'.

Nested elements MAY define their access, but SHALL NOT include privileges in their access defini! tion. Nested elements SHALL inherit their privileges from the next highest element in the model hierarchy. Nested elements that do not define their access SHALL inherit their access from the next highest element in the model hierarchy. For example: A data field in a struct attribute inherits its access qualities from attribute.

Elements SHALL only include the lowest required privilege for a type of access.

That means:

- ¥ An event SHALL define the single privilege required for Read access.
- ¥ A command SHALL define the single privilege required for Invoke access.
- ¥ A readable attribute SHALL define the single privilege required for Read access.
- ¥ A writable (but not readable) attribute SHALL define the single privilege (that is not View or ProxyView) required for Write access.
- ¥ A readable and writable attribute MAY define a single privilege (that is not View or ProxyView) required for both Read and Write access.
- ¥ A readable and writable attribute MAY define the View or ProxyView privilege as required for Read access and one other privilege (that is not View or ProxyView) as required for Write access.

7.6.1. Read Access

Read access means that a request for data values associated with an element SHALL be supported. This quality SHALL only be defined for cluster event and attribute data definitions. This quality SHALL NOT be defined for cluster command definitions.

This quality is implicitly defined for cluster events and does not need to be stipulated explicitly.

7.6.2. Write Access

Write access means that a request to modify attribute data values SHALL be supported. This quality SHALL only be defined for cluster attribute data definitions. This quality SHALL NOT be defined for cluster event and command definitions.

A cluster specification SHALL define the conditions when write access attribute data is not writable, and SHALL define normative or recommended behavior to follow when this occurs.

An implementation that does not support write access for a field with optional write access SHALL have this declared in its product Declaration of Conformity.

7.6.3. Invoke Access

Invoke access means that a request to execute a command SHALL be supported. This quality SHALL

csa connectivity standards alliance Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

only be defined for cluster command definitions, by defining an appropriate privilege level for the command. This quality SHALL NOT be defined for cluster event and attribute data definitions.

7.6.4. Fabric-Scoped Quality

This defines fabric-scoped data that is scoped to an associated fabric .

This quality acts as an additional constraint over those imposed by the existing Read and Write qualities, namely:

- Fabric-scoped attribute data SHALL NOT be writable unless the accessing fabric is the associated fabric of the data.
- A cluster command SHALL NOT alter fabric-scoped data if the associated fabric is not the accessing fabric .

7.6.5. Fabric-Sensitive Quality

This further restricts access to data that is sensitive to the associated fabric .

This quality acts as an additional constraint over those imposed by the fabric-scoped quality, namely:

- Fabric-sensitive data SHALL NOT be readable unless the accessing fabric is the associated fabric of the data. See fabric-scoped data .

Data that is fabric-sensitive SHALL always be fabric-scoped.

7.6.6. View Privilege

An element with the View privilege SHALL support Read (if readable) and Invoke (if invocable) access if the source of the request is granted the View privilege.

A command with the View privilege defined SHALL NOT alter data that is part of its function (e.g. settings, configuration), but MAY alter data that is internal or diagnostic in nature (e.g. usage statistics).

7.6.7. Operate Privilege

An element with the Operate privilege defined SHALL support Read (if readable), Write (if writable), and Invoke (if invocable) access if the source of the request is granted the Operate privilege.

7.6.8. Manage Privilege

An element with the Manage privilege defined SHALL support Read (if readable), Write (if writable), and Invoke (if invocable) access if the source of the request is granted the Manage privilege.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

7.6.9. Administer Privilege

An element with the Administer privilege defined SHALL support Read (if readable), Write (if writable), and Invoke (if invocable) access if the source of the request is granted the Administer privilege.

7.6.10. Timed Interaction

This quality requires the use of a timed interaction.

Timed interactions are used to limit the amount of time an action message is valid and can interact with a node. They are used to prevent a timing attack on the system. For example, a malicious attacker could perform an "intercept, interfere, and replay" procedure whereby a legitimate mes! sage is intercepted, receipt by the intended destination is jammed, and the attacker sends the mes! sage at a later time to cause a malicious action such as unlocking a door at an unintended time. While the practical difficulties of such an attack are high, and the malicious eavesdropper cannot decrypt the action message, the timed interaction provides further mitigation of risk for critical actions.

The timed interaction can be thought of as a 2-phase commit. A precursor action (Timed Request Action) is sent to indicate the valid time window for arrival of some subsequent, primary action. Since the timed request requires a response, an attacker cannot do the store-and-forward timing attack anymore. The lack of an authenticated response from the intended destination will prevent the subsequent primary action from being sent.

A command with this quality SHALL require a timed invoke interaction. A writable attribute with this quality SHALL require a timed write interaction.

An attempted untimed write interaction to a writable attribute with this quality SHALL generate an error response.

An untimed invoke interaction for a request command with this quality SHALL generate an error response.

7.7. Other Qualities

A Quality column defines other qualities not covered in other columns. Some qualities are limited to a specific set of elements. If an element does not have designated qualities, then it SHALL inherit qualities from the next highest element in the model hierarchy. For example: A data field in a struct attribute inherits its access qualities from attribute.

Quality Column	Name	Elements	Description
X	Nullable	data fields	the data type of the data field is nullable
N	Non-Volatile	attribute data	the attribute data value is non-volatile and is persistent across restarts

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Quality Column	Name	Elements	Description
F	Fixed	attribute data	the read only value is a fixed value that never changes, unless the program image changes
S	Scene	attribute	the attribute is part of a scene
P	Reportable	attribute	if best effort reporting is supported then the attribute supports a reporting configuration
C	Changes Omitted	attribute data	fast changing data or data where deltas are meaningless to report, and which will not cause delta changes on subscriptions
I	Singleton	device type	the cluster is a single! ton on the node for the device type
!Q		device type	the quality Q (from those defined above in this table) is disal! lowed.

7.7.1. Nullable Quality

See Nullable .

7.7.2. Non-Volatile Quality

See Persistence .

7.7.3. Fixed Quality

Data with this quality is read only and has a fixed value that never changes, unless the program image changes.

7.7.4. Scene Quality

This quality is supported and described in the Scenes cluster.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

7.7.5. Reportable Quality

The Subscribe interaction supports all attribute data. This quality is supported by other interactions that only require attribute data with this quality to support interval or change reporting.

7.7.6. Changes Omitted Quality

This quality MAY be given to attribute data that is deemed to have a high rate of change or where changes are not meaningful or too large to convey as part of Subscribe interaction .

Attribute data with this quality SHALL support Read Access, but SHALL NOT have delta changes published as part of a Subscribe interaction .

7.7.7. Singleton

Data with this quality, that be indicated by more than one cluster, represents a single instance of the data, for the node.

7.8. Node

A Node encapsulates an addressable, unique resource on the network that has a set of functions and capabilities that a user recognizes distinctly as a functional whole.

This distinction is usually physical, such as the physical device itself, or a logical instance of a physi! cal device.

A node is the highest or outermost first order element in the data model. A node is the outermost unique addressable element of the data model.

A node MAY have multiple node IDs, each ID scoped to a particular fabric. When a node ID is used as the target address of an interaction, the fabric under which the node ID is scoped, is the access! ing fabric for the interaction.

The lower layers in a communication stack supporting this data model SHALL support interactions between nodes on a logical inter-network of nodes. Please see the Interaction Model and System Model specifications that describe relationships and interactions between nodes and data model elements on each node.

It is possible for parts of a node to reside on different processors (e.g. separate application and net! work processors).

A single physical product may support more than one node.

7.9. Endpoint

A node is composed of one or more endpoints. An endpoint is an instance of something that could be a service or virtual device as indicated by a device type .

Each endpoint conforms to one or more device type definitions that define the clusters supported

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

on the endpoint. Clusters are object classes that are instantiated on an endpoint.

The word 'device', depending on the context, may be used as shorthand to denote the device type definition as represented by a device type ID, a device type implementation, or an endpoint (device type instance).

There are also many examples in specification text where 'device' is used, when it would be better, and more accurate to use 'node', 'physical device', or 'product'.

The word 'device' may also be used in cluster specifications to describe application software that is supporting an instance of a cluster server or client. In this case, it would be better, and more accu! rate to use either 'client' or 'server'.

One must be careful to make sure there is no ambiguity when using the word 'device' in specifica! tion text, or better yet, use another word.

7.10. Cluster

Clusters are the functional building block elements of the data model. A cluster specification defines both a client and server side that correspond with each other through interactions. A clus! ter may be considered an interface, service, or object class and is the lowest independent functional element in the data model. Each cluster is defined by a cluster specification that defines elements of a cluster including attributes, events, commands, as well as behavior associated with interactions with these elements. Cluster attributes, events, commands and behaviors are mandatory or optional depending on the definition of the cluster. Optional items may have dependencies.

A cluster specification SHALL list one or more Cluster Identifiers. A Cluster Identifier SHALL refer! ence a single cluster specification and SHALL define conformance to that specification. A cluster instance SHALL be indicated and discovered by a Cluster Identifier on an endpoint. A Cluster Iden! tifier also defines the purpose of the instance.

The server cluster supports attribute data, events and cluster commands. The client cluster initiates interactions, including invocation of cluster commands.

7.10.1. Cluster Revision

The revision of a cluster is to enforce backward and forward compatibility, but still allow clusters to be enhanced, fixed, or updated, without changing the clusterÕs basic function.

A cluster revision SHALL be associated with an approved revision and release of a cluster specifica! tion. The revision of an instance of a cluster SHALL be represented by the global, mandatory, and read only ClusterRevision attribute. Please see ClusterRevision attribute.

Changes to a cluster specification SHALL only augment, not modify the primary function of the cluster. Changes to a cluster specification SHALL be represented by incrementing the cluster revi! sion. New revisions of a client cluster SHALL interoperate with older revisions of the server cluster and vice versa. Interoperability between corresponding cluster instances MAY require reading the cluster revision.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

For example: If a new product client application supporting revision 3 of cluster X wishes to take advantage of the new behavior that is mandated by revision 3, then the application can read the revision of the corresponding server cluster X in each remote endpoint. If a corre! sponding cluster X supports revision 3 or greater, then the behavior is supported.

Examples of changes to a cluster that require incrementing the revision:

¥ Changing the behavior of the cluster

¥ Changing a read only attribute to become writable

¥ Adding new attributes (e.g. min and max of an existing attribute value)

¥ Adding new commands, actions, or behavior

¥ Adding one or more fields to an existing command

¥ Adding a new enumerated value to an attribute

¥ Changing anything that is optional to mandatory

¥ Changing dependencies of optional items

¥ Deprecating parts of the cluster specification

¥ Any non-editorial specification text change

7.10.2. Cluster Optional Features

In general, as the number of optional elements in a cluster specification increases, the number of possible combinations increases, which could decrease the interoperability of that cluster.

Each cluster has a mandatory feature set that consists of mandatory elements such as attributes, commands, fields, values, dependencies, behavior, etc.

A cluster specification MAY have optional feature sets, each supported by a set of elements (see Fea! tureMap).

There is no requirement that each cluster instance supports the same set of optional elements.

If an application knows the ClusterRevision and FeatureMap supported by a cluster instance, then it knows the exact specification text required to be implemented by that instance.

7.10.3. Cluster Data Version

A cluster data version is a metadata increment-only counter value, maintained for each cluster instance. A cluster data version represents an exact & coherent state of cluster attribute data at present. An application may externally hold a data version (called a held data version) published by a cluster instance which then represents a cluster instance state at some time in the past. An appli! cation may use a held data version to optimize future interactions, by indicating the held data ver! sion. A cluster data version is surfaced in the Interaction Model when data is requested. It is used to optimize data read transactions by reducing the need to send the same data. Write interactions may also be qualified with a held data version to disallow changes, unless the cluster instance has the

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

same data version (see Interaction Model). A cluster data version is published as information in some interactions (See Interaction Model). An externally held data version may be included as information in some interactions (See Interaction Model).

A cluster data version SHALL increment or be set (wrap) to zero if incrementing would exceed its maximum value. A cluster data version SHALL be maintained for each cluster instance. A cluster data version SHALL be initialized randomly when it is first published. A cluster data version SHALL be incremented if any attribute data changes.

7.10.4. New Cluster

When considering the creation of a new cluster specification, it is recommended to consider reusing and extending an existing cluster specification. These are the mechanisms to consider, in order, to extend a cluster:

1. Optional elements: attribute data, commands, events, enumerations, etc.
2. Optional feature(s) in the FeatureMap Attribute attribute for a set of elements (see 1)
3. Cluster Aliasing to reuse a cluster specification as a whole, but with a different semantic
4. Cluster Inheritance
5. A new cluster specification

7.10.5. Cluster Aliasing

Cluster aliasing allows the reuse of approved and validated specifications and derived documents, such as test plans, scripts, etc.

- ¥ More than one Cluster Identifier, each with unique purpose and semantic content, MAY map to a single cluster specification.

> For example: A Concentration Measurement cluster specification may be quite abstract but have many mapped Cluster Identifiers each with a more concrete purpose, such as CO_2 or O_2 concentration measurement.

7.10.6. Cluster Inheritance

Cluster inheritance allows the reuse of approved and validated specifications and derived docu! ments, such as test plans, scripts, etc. This allows a new cluster specification to be defined as extending or reducing the requirements of an existing cluster specification, called the base cluster. This also allows an existing cluster specification to be defined as a derived cluster, by creating a new base cluster that is more generic, allowing potential new clusters to be derived from the new base cluster.

- ¥ A derived cluster specification MAY have mandatory requirements that are optional in the base specification.
- ¥ A derived cluster specification MAY remove requirements that are optional in the base specifi! cation.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

- A derived cluster specification MAY remove or make optional a requirement that is mandatory in the base specification, if the resulting specification is deemed useful in its reduced form, and logically a subset of the base clusters.

> For example: The Bridged Device Basic Information cluster is derived as a reduced form of the base Basic Information cluster, where many informational attributes are not mandatory, because the information is not available from devices behind the bridge. However, the derived cluster provides the same, but reduced, function as the base cluster.

- It is RECOMMENDED that an extension or reduction for a derived cluster is one or more features or independent elements, not a modification to custom cluster behavior.
- All new features, elements or behavior introduced by the derived cluster SHALL be defined in the base cluster specification and made optional (in that base cluster specification), to maintain the entire set of requirements and identifier namespace in one place.
- A derived cluster specification SHALL define its own revision (ClusterRevision attribute) that is independent of the base specification.
- A base cluster specification MAY be created from an original base cluster, which then becomes a derived cluster to the newly created base cluster.

If an endpoint supports multiple server clusters that derive or map to the same base cluster specification, then each SHALL represent a single implementation and operate as a single entity or instance. This makes it possible to deploy a new device endpoint with both a base and a derived cluster identifier, which SHALL remain backward compatible to legacy devices that support only the original cluster identifier. Cluster identifiers that are mapped to a single base cluster specification, but are defined for distinctly different purposes, MAY exist together on a device endpoint. If there is no base cluster identifier defined, or no base cluster identifier exists on the same endpoint, then each cluster identifier SHALL represent a separate instance.

It is a good practice to explore the possibility of either deriving a cluster from an existing cluster or creating a base cluster to map or derive new and existing cluster identifiers. See New Cluster for other options.

7.10.7. Status Codes

A cluster specification defines status code responses to actions depending on the cluster instance state. A status code is either a global Interaction Model status code , or a cluster specific status code that is unique to the cluster specification. A global status code is either scoped to the entire action, or to a cluster request path. A cluster specific status code scoped to a cluster instance is indicated by a cluster path. When an interaction defines a Status Response response, the responder SHALL return a global Interaction Model status code. When an interaction response needs to communicate a cluster specific status code, the responder SHALL return the path to the cluster instance, the global status code SUCCESS or FAILURE, and the cluster specific status code. Each cluster specific status code SHALL be associated with either SUCCESS or FAILURE, not both. A cluster specific status code SHALL be, by default, associated with FAILURE unless it is defined as associated with SUCCESS. The global SUCCESS status code means the action was executed for the request path; the global FAILURE status code means that it was not executed.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

- ¥ Cluster-specific status code SHALL be defined using the status type.
- ¥ Cluster-specific status codes MAY have the same numeric values as global status codes. Interac! tion model messages SHALL make it clear whether a particular message field is a global status code or a cluster-specific status code.
- ¥ Cluster-specific status codes SHALL communicate more information than just a generic success or failure condition. Global status codes SHALL be used to communicate such conditions.
- ¥ A server cluster SHALL NOT return a cluster-specific code from another cluster.

7.10.8. Cluster Classification

A cluster SHALL be defined as either a utility cluster or an application cluster.

7.10.8.1. Utility Cluster

A utility cluster is not part of the primary application operation of an endpoint. It may be used for configuration, discovery, addressing, diagnostics, monitoring device health, software update, etc. It may have a temporary relationship with its cluster counterpart.

> Utility cluster examples scoped to an endpoint: Identify, Descriptor, Binding, Groups, etc. Util! ity cluster examples scoped to the node: Basic Information, Diagnostics, etc.

7.10.8.2. Application Cluster

An application cluster supports the primary operation of the endpoint. An application cluster sup! ports one or more persistent application interactions between client and server.

> Example application cluster transactions:
>
> - ¥ On/Off cluster - client sends command to server
> - ¥ Temperature Measurement cluster - server reports data to client

An application cluster is not a utility cluster even though it may support utility functions for itself, such as calibration, modes of operation, etc. An application cluster specification SHALL be agnostic about layers and processes outside of its application domain.

> Example: A particular temperature measurement cluster may exist on different devices, or in different networks, each with different security & commissioning mechanisms and/or poli! cies.
>
> Example: A commissioning clusterÕs domain is commissioning, but not temperature measure! ment.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

7.11. Command

A cluster command is a set of data fields , each of a data type that is conveyed between client and server cluster instances to invoke a behavior on the receiver of the command.

Each command SHALL be listed in a table with data quality columns: ID, Name, Direction, Response, Access, Conformance.

The command table SHALL define the direction of the command as either client to server or server to client. The command table SHALL define the access privileges for each request command or omit the privileges for the default (see default access privileges). The command table SHALL NOT define privileges for a response command. The command table SHALL define a possible response to the command, if any. The command table SHALL define conformance for each command.

A command that is not a response (in the Response column) is a request command. Conformance for a client to server command means the server SHALL recognize and support the client to server command and generate responses as defined. Conformance for a server to client command means the server SHALL send the command as cluster behavior defines, such as in response to a client to server command. Conformance for a command can depend on supported server features. A client SHALL NOT be required to support optional commands or commands depending on an optional feature.

A command description SHALL define when a command is generated. A command description SHALL define the effect upon receipt of a command which may be:

¥ a response command

¥ a success status response

¥ an error status response

¥ no response

A command definition SHALL clearly define any side-effects on fabric-scoped data, if applicable.

A command is identified and indicated with a command ID that SHALL be unique to the cluster *.

*Note

Some legacy clusters have reused the same command ID twice to indicate one command from the client and another from the server. Moving forward, command IDs SHALL NOT be reused in that fashion.

A cluster command table SHALL have a Response column with the following values:

Response Column	Description
N	no response is returned for this command
Y	status only is returned for this command

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Response Column	Description
command	name of the response command to this com! mand

A cluster command table SHALL have a Direction column with the following values:

Direction Column	Description
client (server	command is conveyed from the client to the server cluster
client) server	command is conveyed from the server to the client cluster

Each command SHALL be described in its own section with a table defining command fields (if any).

7.11.1. Command Fields

A command MAY indicate zero or more fields that are defined in a table. Each command field is defined as a row in the table with these columns:

Column	Description
ID	This is the unique field ID of the field
Name	This is the unique name of the field
Type	This is the data type of the field
Constraint	see Constraint
Quality	see qualities
Default	see Default
Conformance	see Conformance

Command field conformance defines the sender requirements to include the field in a well-formed command for the revision of the cluster. A new command field or a newly made-mandatory com! mand field in a newly revised cluster specification may be omitted by a legacy sender. The cluster specification shall define clear behavior upon receipt of any possible well-formed command with fields that are not present. The cluster specification shall take into consideration the revision his! tory of possible well-formed commands from legacy implementations. To allow deprecation, it is recommended that command fields have a well-defined default value (such as null), and associated default behavior, that is equivalent to omitting the field. Well-defined behavior, for a field that is not present, may be no behavior at all.

For example

A newly revised Noise cluster adds a new mandatory Volume field to the MakeNoise com! mand. Legacy receivers will ignore the Volume field, and legacy senders will not include the field.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Another example

The Volume field is mandatory for the original cluster and there is a proposal to make it optional. The Volume field null value has the semantic of ignoring the field, so instead of making it optional, the default value is used. This would make the receiver logic simpler.

7.12. Attribute

An attribute is cluster data. Each attribute SHALL be listed in a table with data quality columns: ID, Name, (Data) Type, Constraint, other Quality, Access, Default (value), and Conformance. An attribute SHALL also define its associated semantics and behavior. Attributes reflect queryable/set! table state, configuration and capabilities of a device. If no privileges are explicitly defined for an attribute, then default access privileges take effect. Attribute data MAY also have these other quali! ties:

Quality	Short	Description
Scene	S	indicates that the data is part of a scene
Persistent	N	indicates that the data value is persistent across restarts
Fixed	F	indicates that the read only data value will never change
Nullable	X	indicates that the data may have a value of null

Fabric-scoped attribute data SHALL be defined as a fabric-scoped list .

7.12.1. Persistence

Persistent data retains its value across a restart.

A restart is:

- ¥ a program restart or reboot
- ¥ power cycle reboot
- ¥ user-initiated reboot
- ¥ reboot initiated from a program image upgrade

A factory reset is not such a restart. A factory reset is a deliberate behavior to reset persistent data back to its original state when the product left the factory.

Cluster attributes that represent configuration data SHALL be persistent data unless otherwise specified.

For example: a writable attribute that persistently changes the behavior (or mode) of the clus!

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

ter.

Examples of non-configuration data: device state data, data that is calculated or comes from an external source, such as a sensor value, a time value, etc.

Many clusters define persistent data that is not surfaced as attributes, but is managed by com! mands. Commissioning or configuration data that is created to allow the cluster to perform its func! tion is persistent data. A group table entry and binding entries are both persistent data across a restart.

When a persistent cluster attribute represents controlled state of the device, the device SHALL restore the attribute value to the value before the restart was initiated, and put the device in the state that is represented by the restored value.

For example: After an OTA cluster restart, clusters that have visible state attributes, such as the state of a light, or a window shade SHALL be persistent and define these attributes as per! sistent.

Some cluster specifications add a dependency with a persistent configuration attribute A that con! tains a value to use to restore persistent state attribute B after a restart. This is perfectly valid but cluster specific.

Cluster state data that is not controlled, such as sensor data, is not considered persistent.

The cluster specification may put dependencies and limitations on persistent data.

7.13. Global Elements

Below is a list of global elements. These are used for self-description of the server.

ID	Name	Element	Type	Con! straint	Quality	Access	Default	Confor! mance
0xFFFD	Cluster! Revision	attribute	uint16	1 to max	F	R V		M
0xFFFC	Fea! tureMap	attribute	map32		F	R V	0	M
0xFFFB	Attribut! eList	attribute	list[attrib- id]		F	R V		M
0xFFFA	EventList	attribute	list[event- id]		F	R V		P, M
0xFFF9	Accept! edCom! mandList	attribute	list[com! mand-id]		F	R V		M

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

ID	Name	Element	Type	Con! straint	Quality	Access	Default	Confor! mance
0xFFF8	Generat! edCom! mandList	attribute	list[com! mand-id]		F	R V		M
0xFE	FabricIn! dex	struct or event field	fabric-idx	1 to 254		R V F		fabric-scoped

7.13.1. ClusterRevision Attribute

The ClusterRevision attribute indicates the revision of the server cluster specification supported by the cluster instance. An implementation of a cluster specification before the ClusterRevision attribute was added SHALL have an assumed cluster revision of 0 (zero). For a new cluster specifi! cation, the initial value for the ClusterRevision attribute SHALL be 1 (not zero).

A history of revision numbers for a cluster specification release is listed in the Revision History sec! tion for a cluster specification. Each new revision of a cluster specification SHALL specify a new revision number incremented (by 1) from the last. The highest revision number in a cluster specifi! cationÕs Revision History is the revision number for the cluster specification. Therefore, a Cluster! Revision attribute value SHALL be the (highest) revision number of the cluster specification that has been implemented.

7.13.2. FeatureMap Attribute

Each instance of a cluster SHALL support this attribute.

The FeatureMap attribute SHALL indicate whether the server supports zero or more optional clus! ter features. A cluster feature is a set of cluster elements that are mandatory or optional for a defined feature of the cluster. If a cluster feature is supported by the cluster instance, then the cor! responding bit SHALL be set to 1, otherwise the bit SHALL be set to 0 (zero). All undefined bits in this attribute SHALL be set to 0 (zero).

The set of cluster elements that are designated as mandatory (M) are implicitly part of the manda! tory cluster feature set, and do not have a bit in the FeatureMap attribute.

A cluster specification SHALL support this attribute if the cluster supports features. If a cluster specification is revised to support features (and so this attribute), each bit in the FeatureMap attribute SHALL have a defined default value (1 or 0), to conform with the previous revision of the cluster specification, that did not support the FeatureMap attribute. The value of 1 means the fea! ture elements were mandatory (M) in the previous revision. The value of 0 (zero) means the ele! ments were optional in the previous revision.

Any newly created feature set of a cluster SHALL be dependent on that cluster.

Feature sets are revision controlled as part of a cluster using the ClusterRevision attribute. The clus! ter specification is the independent element that is revision controlled. A remote application read! ing the FeatureMap Attribute and ClusterRevision Attribute will then know exactly what features

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

are supported in the cluster instance.

Each feature set SHALL be well defined within the cluster specification. Each feature SHALL be mapped to a short capitalized code name for the feature set to be referenced as a conformance tag in the cluster specification text, including the Conformance columns defining the elements of the cluster.

If a cluster defines more than 32 feature sets, then it will be necessary to add another feature bitmap attribute. Any client trying to reference the new feature set will know about the new bitmap, because it knows about the new feature set(s). Legacy products will not know about the new feature set nor the new bitmap.

For a cluster whose definition which does not define a FeatureMap, the server SHALL set this attribute to 0 (zero).

7.13.3. AttributeList Attribute

Each instance of a cluster SHALL support this attribute. This attribute SHALL be a list of the attribute IDs of the attributes supported by the cluster instance.

7.13.4. AcceptedCommandList Attribute

This attribute is a list of client generated commands which are supported by this cluster server instance.

Each instance of a cluster SHALL support this attribute.

This attribute SHALL be a list of the command IDs for client generated commands that are sup! ported and processed by the server.

For each client request command in this list that mandates a response from the server, the response command SHALL be indicated in the GeneratedCommandList attribute.

7.13.5. GeneratedCommandList Attribute

This attribute is a list of server generated commands. A server generated command is a server to client command.

Each instance of a cluster SHALL support this attribute.

This attribute SHALL be a list of the command IDs for server generated commands.

For each command in this list that is a response to a client command request, the request command SHALL be indicated in the AcceptedCommandList attribute.

7.13.6. EventList Attribute

Each instance of a cluster SHALL support this attribute. This attribute SHALL be a list of the event IDs of the events supported by the cluster instance.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

NOTE	Support for EventList attribute is provisional.

7.13.7. FabricIndex Field

This field SHALL be present for fabric-scoped data . This field does not have to be defined explicitly in the field table for fabric-scoped data.

This field SHALL NOT be present in a write interaction. For a write interaction, the server SHALL provide the value of the accessing fabric-index as the FabricIndex field value to processing logic, after receipt of the interaction. For a read interaction this field SHALL be included in all reported data that is defined as fabric-scoped.

7.14. Event

An event defines a record of something that occurred in the past. In this regard, an event record can be thought of as a log entry, with an event record stream providing a chronological view of the events on the node.

Unlike attributes, which do not provide any edge-preserving capabilities (i.e. no guarantees that every attribute change will be conveyed to observers), events permit capturing every single edge or change and conveying it reliably to an observer. This is critical for safety and security applications that rely upon such guarantees for correct behavior.

Each cluster event is listed in a table that defines: ID, Priority , Access, Conformance .

Event records are readable , and do not require the read access quality to be explicitly defined.

7.14.1. Priority

Each event record has an associated priority . This priority describes the usage semantics of the event.

The following table defines possible event priorities:

Priority	Description
DEBUG	For engineering debugging/troubleshooting
INFO	Events that either drive customer facing fea! tures or provide insights into device functions that are used to drive analytics use-cases
CRITICAL	Events that impact physical safety of users, or ongoing reliable operation of the node function (or cluster of the node)

7.14.2. Event Record

An event record is created by the node at the time the event happens. That record SHALL have the following data fields associated with it that are common to all events:

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

- ¥ Event Number
- ¥ Timestamp
- ¥ Priority

Each generated event record SHALL have an event priority that MAY override the defined priority for that event.

Each event SHALL be described in a section that defines the purpose of the event and data fields of the event (if any). Event fields SHALL be defined in the form of a struct in a table with the following columns: ID, Name, Type, Constraint, Quality, Default, Conformance.

7.14.2.1. Event Number

This is an event number value that is scoped to the node. This number SHALL be monotonically increasing for the life of the node. This monotonicity guarantee SHALL be preserved across restarts .

Between restarts, each event record SHALL be assigned a number that is exactly 1 greater than the last created event record on that Node.

When a node restarts, the event number MAY increase by a larger step than 1. Rationale : Nodes do not need to write every new value of the event number counter to permanent storage each time it is increased (e.g. to prevent flash wear due to many write operations). One example strategy to achieve reduction of non-volatile storage updates is described below:

1. Read the counter value at start-up.
2. Before processing any message, write counter + N to storage, where N is a carefully chosen number (e.g. 1000). This number N should be chosen carefully in order not to exhaust the life! time 64-bit counter space.
3. Process messages normally until the counter has a value one less than the counter in storage. When this happens, store counter + N to storage.

7.14.2.2. Timestamp

Each event record SHALL have a timestamp at the time it was created (and not when it is reported to a client). This timestamp SHALL either be System Time in Microseconds or Epoch Time Microsec! onds .

7.14.3. Buffering

Event records SHALL be buffered on the Node, with priority given to events of a higher priority level over a lower priority level. Within a priority level, newer event records SHALL overwrite older event records. The Node SHOULD only overwrite older events if there are newer events cre! ated and there is insufficient space to retain both.

7.14.4. Event Filtering

Interactions that report event records MAY be filtered by event ID and/or event number .

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

7.14.5. Fabric-Sensitive Event

An entire event MAY be defined as having the fabric-sensitive quality ; otherwise, it SHALL NOT be associated with a fabric.

A read interaction SHALL NOT filter event records, based on fabric, for event records that are not associated with a fabric.

A read interaction SHALL NOT report fabric-sensitive event records that are associated with a fab! ric different than the accessing fabric .

A fabric-sensitive event SHALL include the global FabricIndex field . For a fabric-sensitive event it is not required to define the FabricIndex field in the event field table.

7.15. Device Type

In this architecture model, a device type is the highest semantic element. A device type defines con! formance for a set of one or more endpoints. A device type defines a set of requirements for the node or endpoint in the market.

A device type SHALL define the cluster support for an endpoint. A composed device type MAY define one or more other device types as part of the composed device type.

A device type definition MAY define or use predefined conditions from requirements, limitations and/or capabilities of the node. A device type definition MAY define or use predefined conditions on one or more underlying stack standard(s).

A device type MAY define support of a cluster as dependent upon a condition. A device type defini! tion MAY specify optional clusters that are recommended as enhancements.

A device type definition MAY refine cluster conformance:

- ¥ Support of optional cluster elements or features MAY be changed to mandatory depending on device type conditions.
- ¥ Support of optional cluster elements or features MAY depend on device type conditions.

A device type definition SHALL specify a device type ID, device revision, and a set of one or more mandatory clusters including each clusterÕs minimum revision.

A device type definition MAY be generic and allow many similar clusters, where at least one instance SHALL be required.

> For example: a simple sensor device.

If all sensor devices are common in cluster requirements (except the clusters that perform the sens! ing), then there is no reason to create a device type for each sensor cluster.

A device type definition MAY be very specific and list particular clusters as mandatory.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

For example: a door lock device or thermostat.

7.15.1. Device Type Revision

A device type revision is an unsigned integer that is associated with an approved revision and release of a device type definition. The initial value for a device type revision SHALL be 1. The ini! tial revision (1) of a device type definition SHALL require the latest (at the time of definition the cluster) certifiable revisions of the clusters it mandates. Device type implementations MAY support later revisions of the mandatory clusters as they become certifiable. Any mandatory changes to the device type definition SHALL only augment, not modify, the function of the device. Any changes SHALL increment the version of the device. Newer versions of the device SHALL interoperate with older revisions at the older revisionÕs level of functionality.

Examples of changes to a device type definition that require incrementing the revision:

- ¥ Mandating a higher revision of one or more mandatory clusters
- ¥ Changing an item from optional to mandatory
- ¥ Deprecating parts of the device type definition

7.15.2. Device Type Composition

A device type definition MAY be a composed device type and therefore require other device types for its composition. A device type instance MAY be composed of other endpoints that support extra cluster instances. Please see the System Model specification for more details.

7.15.3. Device Type Classification

Each device type definition SHALL specify the endpoint as being either a Utility, or Application. Each device type definition SHALL specify the scope as either endpoint or node. Each Application device type definition SHALL specify the endpoint as being either Simple or Dynamic.

7.15.3.1. Utility Device Type

A Utility device type supports configuration and settings. A utility device type definition SHALL define requirements for utility clusters. A utility device type MAY also represent the physical device or product. There MAY be more than one endpoint supporting a utility device type on a node. Example utility cluster categories: OTA upgrade, diagnostics, basic information. Example utility device type categories: bridge, proxy, power source.

7.15.3.2. Application Device Type

Application devices types are typically the most common endpoints on a node and in the network. An endpoint supporting an application device type is an application endpoint. An Application device type SHALL be scoped to the endpoint. An application endpoint SHALL support clusters the primary application function of the endpoint. Application category examples: HVAC, lighting, home

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

security, etc.

7.15.3.3. Simple Device Type

A Simple device type supports local control that is persistent, independent, and unsupervised. A Simple device type is an Application device type. Simple devices types are typically the most com! mon endpoints in the network. Simple device type examples: sensors, actuators, lights, on/off switches, on/off power outlets, etc. Simple endpoints support independent operation without cen! tral control or gateways. An endpoint supporting a simple device type is a simple endpoint. Simple endpoints SHALL support relationships through bindings.

7.15.3.4. Dynamic Device Type

A Dynamic device type supports intelligent and supervisory services, such as commissioning, moni! toring, trend analysis, scheduling and central management. A dynamic device type is an application device type. An endpoint supporting a dynamic device type is a dynamic endpoint. A dynamic end! point is typically found on a central controller where there exists an intelligent supervisory applica! tion that manages simple device control applications. Typically, a dynamic endpoint supports client clusters for central control, management, monitoring or supervisory functions. Typically, the prod! uct supporting a dynamic endpoint has visibility into the entire network (or part thereof) of simple endpoints.

A dynamic endpoint client cluster instance MAY be used to multiplex transactions to or from multi! ple simple device server clusters in the network. A dynamic endpoint client cluster MAY initiate interactions to many server clusters in the network. A dynamic endpoint client cluster MAY receive data from many server clusters in the network. Dynamic endpoints MAY support relationships through bindings. A dynamic device endpoint MAY support one or more external agents, outside the node stack, that manage relationships. External agents include, but are not limited to, a cloud application, a smartphone, an in-home display, or a configuration tool.

7.15.3.5. Device Type Scope

A node device type is a utility device type scoped to a node. A node device type definition SHALL support clusters that represent the entire node. An endpoint supporting a node device type is a node endpoint. A node endpoint MAY also represent the physical device or product. There MAY be more than one node endpoint on a node.

Other classes of device types are endpoint scoped device types.

7.15.4. Extra Clusters on an Endpoint

An endpoint MAY support later revisions of a cluster mandated by the device type definition. An endpoint MAY support extra clusters not mandated by the device type definition. An endpoint MAY support optional features or cluster items (attributes, commands, events, etc.), that are not man! dated by the device type definition. Extra clusters, features, or cluster items, SHALL only augment, not modify, the function of the device type or clusters.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

7.16. Non-Standard

This architecture model provides mechanisms for non-standard or manufacturer specific items such as clusters, commands, events, attributes and attribute data fields. These items MAY be sup! ported on a certified product. Such vendor specific items SHALL NOT change the standard behavior of the standard items. The specific function of a vendor specific item cannot be tested as part of cer! tification. They can only be tested to verify that they do no harm, and conform to proper behavior with regard to identification, discovery, error processing, etc. A non-standard item SHOULD NOT take the place of a standard item that provides the same function. It is up to the certification authority to make a judgment call that is in keeping with the spirit of these requirements. Imple! menters are encouraged to develop and certify standard items, not non-standard items.

7.17. Data Field

A data field is any attribute, field or entry that is not a collection data type, or data that is not sur! faced as an attribute, but defined in a cluster specification.

Optional attribute data MAY be referenced as data fields in other attribute specifications within the same cluster specification. Cluster specifications also define data fields that are not surfaced, such as temporary calculated values, or persistent state values. Any defined data value in a cluster speci! fication is a data field.

Each cluster data field SHALL be defined with a table including these columns for data qualities:

- ¥ Data Type
- ¥ Constraint
- ¥ Quality
- ¥ Access
- ¥ Default
- ¥ Conformance

A data field SHALL inherit (if possible) the qualities from the cluster first-order element of which it is part, unless overridden. It SHOULD be rare to override inherited qualities.

> For example: If an attribute is a struct data type, that is readable and writable, then all fields of the struct are readable and writable.

New or optional data fields MAY not be recognized by a receiver, such as a legacy receiver. The data field description SHALL define default behavior (such as absence of behavior) when a new or optional data field is not present. It is recommended to define or use a feature when adding new or optional data fields, to better indicate conformance. It is recommended to define a default value , such as a null value, that indicates such default behavior.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

7.17.1. Nullable

When a data field value is required to designate an unknown, invalid, or undefined data value, and there is no obvious data value (e.g. zero), that is within the valid range to indicate this, the data field MAY be designated as nullable, so that an implemented instance of data MAY have the value of null.

In this context, these wordings have the same meaning:

- ¥ The data field has the value of null.
- ¥ The data field has the null value.
- ¥ The data field is the null value.
- ¥ The data field is null.

Representation of null for the implementation of the Data Model is a consideration of the underly! ing encoding specification. The encoding layer SHALL have the capability to indicate null for any nullable data field. How the encoding layer indicates null is outside the scope of the Data Model specification.

All data fields MAY be defined to be nullable, regardless of data type.

A cluster specification SHALL define whether a data field is nullable. A cluster specification SHALL define the meaning of the null value.

Composite data types that have a length (i.e. octet string and list), and derived types that have those as the base type, SHALL NOT differentiate semantically between the null value and the empty (zero length) value. In particular a zero-length value SHALL be allowed for nullable values of these types no matter what other length constraints are imposed on the value, and SHALL have the same semantics as the null value.

7.17.2. Optional or Deprecated

An optional or deprecated data field that is not implemented, and therefore does not exist, SHALL NOT be indicated as the null value. How the encoding layer encodes non-existent data is outside the scope of the Data Model specification.

The Conformance column shall define if a data field is optional or deprecated. To manage the data identifier namespace, a deprecated data field SHALL NOT be removed from text that lists its identi! fier and default value. The description text of a deprecated data field SHALL be removed for new revisions of specification text.

If the specification text of a cluster depends on the value of an optional or deprecated data field of the same cluster, then the data field SHALL have a well-defined default value that SHALL be used when the data field is not implemented.

7.17.3. Constraint & Value

The tables below describe the nomenclature for describing constraints and default data values. This nomenclature is used in the cluster specifications for data value constraints, defaults, and other definitions.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

7.17.3.1. Common Literal Values

These values are commonly used in cluster text and Default columns in cluster data definition tables.

Value	Description
0	The numeral zero is used to indicate the zero value for analog data. This is equivalent to the boolean value FALSE.
1	This is used for any analog data type to mean that the value is 1. This is equivalent to the boolean value TRUE.
FALSE	"FALSE", "false" or "False" is a boolean value and is equivalent to 0 (zero).
TRUE	"TRUE", "true" or "True" is a boolean value and is equivalent to 1.
NaN	Not a Number defined for any floating point val! ues.
null	This indicates the value of null.
empty	This indicates empty list or string data.
min	The minimum possible data value for the data type.
max	The maximum possible data value for the data type.
numeric units	Some number in some well-defined units as described in the data type (e.g. 100 ^{o} C)

7.17.3.2. Constraint

The Constraint column is valid for any attribute or data field of an attribute, event, command or struct. It is RECOMMENDED to always define a constraint for any data field.

Constraint	Description
desc	Defines the constraint is defined in the descrip! tion section
Numeric Data Type Constraints	
x*	Defines a value that is supported.
x to y	Defines a supported value range.
max y	Defines the value range from min to y
min x	Defines the value range from x to max.
all	Defines that all values are supported. Same as "min to max".

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Constraint	Description
constraint , constraint É	Defines support of a union of two or more value and value range constraints
Octet String Data Type Constraints	
x	Defines the size range in bytes to be exactly x
x to y	Defines the size range in bytes from x to y
min x	Defines that the size limit supported is a mini! mum of x bytes
max y	Defines the size limit supported is a maximum of x bytes
all	Defines no constraint on size of the string. Same as "min to max".
constraint , constraint É	Defines support of a union of two or more size range or size limit constraints
List Data Type Constraints	
x	Defines the range of entries to be exactly x
x to y	Defines the range of entries from x to y
min x	defines the limit supported is a minimum of x entries
max y	Defines the limit supported is a maximum of x entries
all	Defines no constraint on the number of entries in the list. Same as "min to max".
constraint , constraint É	Defines support of a union of two or more list range or limit constraints
list_constraint [entry_constraint]	Defines list_constraint as a list constraint and entry_constraint as a constraint on the entry data type. See also list entry qualities
Character String Data Type Constraints	
char_constraint [z]	Defines char_constraint as the string constraint in bytes and z as the maximum number of Uni! code codepoints.

* x, y, or z are literal values of the data type or from the Common Literal Values .

7.17.3.3. List and String Constraint

The minimum number of entries for list or size of a string SHALL be 0 (zero), unless redefined using the above notation.

A comma delimited set of constraints for a list or string defines a union constraint. A union con!

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

straint SHALL only have one minimum (min x) constraint and one maximum (max y) constraint. A union constraint SHALL NOT define a range below the minimum constraint or range greater than the maximum constraint, including the defined minimum (min) and maximum (max) for the data type.

A constraint on a list or string data means that the data SHALL always be indicated within that con! straint. A constraint on a writeable list or string data means that the data SHALL support writing within the constraint, and SHALL NOT support writing outside the constraint.

7.17.3.4. Read Only vs Write Access

7.17.3.5. Effective Maximum for Character String Data Type

A server SHALL support up to the maximum in char_constraint for a character string data type. The character string data SHALL NOT contain more than z Unicode codepoints.

> Example: A string with a constraint of "max 128 [32]" dictates that the server provide for a 128 byte string, but the string may contain up to 32 Unicode codepoints

7.17.3.6. Nullable in Range

If data is nullable then null SHALL be a valid value.

If the data type is a list or derived from a list, and the list is nullable, then a length of 0 (zero) SHALL be supported, and defined in the constraint column.

If the data type is an octet string, or derived from an octet string (e.g. character string), and the data is nullable, then a length of 0 (zero) SHALL be supported, and defined in the constraint column.

7.17.4. Default Column

A default value defined in the Default column is not meant to be the value used when the server returns to factory fresh settings. Specified conformance for data fields may be optional or change over time. A default value is defined to complete dependencies when the actual data field value is not present.

A data field SHALL have a defined default value when:

- ¥ the data field is new, and a default is required for backwards compatibility with legacy instances
- ¥ the data field is optional, deprecated, or obsolete and therefore is not always present
- ¥ an initial value is needed before the application starts
- ¥ the value cannot be determined by the application for the instance
- ¥ there is a dependency on the attribute value to formulate other data or affect behavior

If a default value is not defined for a data field, the default value is determined by the following conditions:

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

¥ If the data field is nullable then the default value SHALL be null

¥ Else the default value SHALL be one of the following, depending on type:

%Boolean: false

%Analog: 0 or 0.0, depending on range

%Bitmaps: 0

%Enumeration: MS

%Composite:

& String: empty

& List: empty

& Struct: default is recursively composited from the defaults of its member fields

%Derived types: use the default value of the base type

These are the options for the Default column used for attributes or attribute, command or event data:

Default Column	Description
x	a literal value x of the data type, or as defined in Common Literal Values
MS	a manufacturer or implementation specific value

If the default value of a data field is specified as manufacturer specific, then there SHALL be no defined default value and the application SHALL support a manufacturer specific value that is in the valid range.

7.18. Data Types

Each data field in a cluster specification SHALL have a well-defined data type. Each attribute in a cluster specification SHALL map to a single data type.

The table indicates for each data type whether it defines an analog or discrete value. Values of ana! log types MAY be added to or subtracted from other values of the same type and are typically used to measure the value of physical properties that can vary continuously over a range. Values of dis! crete data types only have meaning as individual values and SHALL NOT be added or subtracted.

Some data types specify bit-widths for future potential growth in range (analog) or number of val! ues (discrete).

Cluster specifications SHALL use the unique data type short name to reduce the text size of the specification.

7.18.1. Base Data Types

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Class	Data Type	Short	ID	Size
Discrete	Boolean			
	Boolean	bool	0x10	1 byte
	Bitmap			
	8-bit bitmap	map8	0x18	1 byte
	16-bit bitmap	map16	0x19	2 bytes
	32-bit bitmap	map32	0x1B	4 bytes
	64-bit bitmap	map64	0x1F	8 bytes

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Class	Data Type	Short	ID	Size
Analog	Unsigned Integer			
	Unsigned 8-bit integer	uint8	0x20	1 byte
	Unsigned 16-bit integer	uint16	0x21	2 bytes
	Unsigned 24-bit integer	uint24	0x22	3 bytes
	Unsigned 32-bit integer	uint32	0x23	4 bytes
	Unsigned 40-bit integer	uint40	0x24	5 bytes
	Unsigned 48-bit integer	uint48	0x25	6 bytes
	Unsigned 56-bit integer	uint56	0x26	7 bytes
	Unsigned 64-bit integer	uint64	0x27	8 bytes
	Signed Integer			
	Signed 8-bit inte! ger	int8	0x28	1 byte
	Signed 16-bit inte! ger	int16	0x29	2 bytes
	Signed 24-bit inte! ger	int24	0x2A	3 bytes
	Signed 32-bit inte! ger	int32	0x2B	4 bytes
	Signed 40-bit inte! ger	int40	0x2C	5 bytes
	Signed 48-bit inte! ger	int48	0x2D	6 bytes
	Signed 56-bit inte! ger	int56	0x2E	7 bytes
	Signed 64-bit inte! ger	int64	0x2F	8 bytes
Analog	Floating Point			
	Single precision	single	0x39	4 bytes
	Double precision	double	0x3A	8 bytes

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Class	Data Type	Short	ID	Size
Composite	String			
	Octet string	octstr	0x41	desc
	Collection			
	List	list	0x48	desc
	Struct	struct	0x4C	desc

7.18.1.1. Boolean

The Boolean type represents a logical value, either FALSE or TRUE.

- ¥ FALSE SHALL be equivalent to the value 0 (zero).
- ¥ TRUE SHALL be equivalent to the value 1 (one).

7.18.1.2. Bitmap (8, 16, 32 and 64-bit)

This data type is typically used to represent simple cluster settings or state that are treated as whole.

The Reserved Bit Fields conventions define reserved bitmap data.

- ¥ It is RECOMMENDED to define more bits than initially needed to be able to support more values for later revisions.
- ¥ The Bitmap type MAY be used to support up to 8, 16, 32 or 64 boolean values.
- ¥ Bits MAY be combined to enumerate other values.
- ¥ Bits SHOULD be combined as contiguous bit fields.
- ¥ Future revisions MAY require non-contiguous bit fields.
- ¥ The conformance for a bit in a bitmap SHALL be mandatory or dependent upon an existing dis! coverable element, and therefore SHALL NOT be purely optional.

Allowable Conformance for a bit in a bitmap:

- ¥ Mandatory
- ¥ Dependent upon a Feature supported in the FeatureMap attribute.
- ¥ Dependent upon the support of an attribute.

A nullable bitmap SHALL NOT permit use of the most significant bit.

7.18.1.3. Unsigned Integer (8, 16, 24, 32, 40, 48, 56 and 64-bit)

This type represents an unsigned integer with length of N bits and a usable range of:

- ¥ $[0..2^N-1]$ if not nullable OR
- ¥ $[0..2^N-2]$ if nullable.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

The following table presents the representable values following the above rules:

Width N (bits)	Minimum value	Maximum value if nullable	Maximum value if not nullable
8	0 (0x00)	254 (0xFE)	255 (0xFF)
16	0 (0x0000)	65534 (0xFFFE)	65535 (0xFFFF)
24	0 (0x000000)	16777214 (0xFFFFFE)	16777215 (0xFFFFFF)
32	0 (0x00000000)	4294967294 (0xFFFFFFFE)	4294967295 (0xFFFFFFFF)
40	0 (0x0000000000)	1099511627774 (0xFFFFFFFFFE)	1099511627775 (0xFFFFFFFFFF)
48	0 (0x000000000000)	281474976710654 (0xFFFFFFFFFFFE)	281474976710655 (0xFFFFFFFFFFFF)
56	0 (0x00000000000000)	72057594037927934 (0xFFFFFFFFFFFFFE)	72057594037927935 (0xFFFFFFFFFFFFFF)
64	0 (0x0000000000000000)	18446744073709551614 (0xFFFFFFFFFFFFFFFE)	18446744073709551615 (0xFFFFFFFFFFFFFFFF)

7.18.1.4. Signed Integer (8, 16, 24, 32, 40, 48, 56 and 64-bit)

This type represents an signed integer with length of N bits and a usable range of:

- ¥ $[-(2^{(N-1)})..2^{(N-1)}-1]$ if not nullable OR
- ¥ $[-(2^{(N-1)}-1)..2^{(N-1)}-1]$ if nullable.

Whether to use twoÕs complement or another representation for the implementation of the Data Model is a consideration of the underlying encoding specification.

The following table presents the representable values in base-10 following the above rules:

Width N (bits)	Minimum value if nullable	Minimum value if not nullable	Maximum value
8	-127	-128	127
16	-32767	-32768	32767
24	-8388607	-8388608	8388607
32	-2147483647	-2147483648	2147483647
40	-549755813887	-549755813888	549755813887
48	-140737488355327	-140737488355328	140737488355327
56	-36028797018963967	-36028797018963968	36028797018963967

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Width N (bits)	Minimum value if nullable	Minimum value if not nullable	Maximum value
64	-9223372036854775807	-9223372036854775808	9223372036854775807

7.18.1.5. Enumeration (8-bit, 16-bit)

This data type employs scalars to represent context-specific values available from an enumerated set. This data type is nullable.

External standards may be referenced as well as listing the values for the external standard. If the external standard adds values after a specification is adopted, those new values are allowed, but optional. Enumeration values are defined in a table with a Conformance column. When the defini! tion of an enumeration is missing a Conformance column, all values SHALL be considered to have mandatory conformance.

All mandatory readable enumeration values SHALL be understood by the client. All mandatory writable enumeration values SHALL be understood by the server.

If a client indicates an enumeration value to the server, that is not supported by the server, because it is optional, deprecated, or a new value unrecognized by a legacy server, then the server SHALL generate a general constraint error, unless the cluster defines alternate behavior, such as:

- ¥ convert the value to a mandatory value
- ¥ ignore the value
- ¥ generate a cluster specific error

With regard to revising a cluster specification:

- ¥ It is RECOMMENDED that a client be as strict as possible by indicating only values that a server supports.
- ¥ It is RECOMMENDED that the server be as forgiving as possible when processing unsupported values.

Note that indicated enumerations MAY comprise only a strict subset of the required enumerations.

> For example: If a server implementation can never enter an enumerated state XYZ, then the value XYZ would never be indicated, therefore the server would not have to support XYZ.

7.18.1.6. Single-Precision

The single precision number format is based on the IEEEÊ754-2019single precision (32-bit) format for binary floating-point arithmetic.

See IEEEÊ754-2019for more details on the representable values.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

7.18.1.7. Double Precision

The double precision number format is based on the IEEEÊ754-2019 double precision (64-bit) format for binary floating-point arithmetic.

The format and interpretation of values of this data type follow the same rules as given for the sin! gle precision data type, but with wider mantissa and exponent ranges.

See IEEEÊ754-2019 for more details on the representable values.

7.18.1.8. Octet String

The octet string data type defines a sequence of octets with a finite octet count from 0 to 65534. It is RECOMMENDED to define a constraint on the maximum possible count.

7.18.1.9. List

A list is defined as a collection of entries of the same data type, with a finite count from 0 to 65534. A cluster specification may define further constraints on the maximum possible count. The list entry data type SHALL be any defined data type, except a list data type, or any data type derived from a list.

The quality columns for a list definition are for the list.

The list entries are indicated with an index that is an unsigned integer starting at 0 (zero). The maintained order of entries, by index, is defined in the cluster specification, or undefined. Data that is defined as a list is indicated with "list[X]" where X is the entry type. The data type of the list entry has its own qualities, constraints, and conformance.

To define qualities for the list entry data type, make the list entry data type a defined local derived data type, with a table including the columns required to define and constrain the data type.

For example: Derived data types defined here:

Name	Type	Constraint	Quality	É
Month! NameString	string	3	F	É
MonthNumber	uint8	1 to 12		É

SummerStruct defined here:

ID	Name	Type	Constraint	Quality	É
0	Year	int16	-1000 to 3000		É
1	Summer! Months	list[Month! Number]	max 12	N	É

Used Here:

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

ID	Name	Type	Constraint	Quality	É
0	MonthNames	list[Month! NameString]	12	N	É
1	SummerYears	list[Summer! Struct]	max 50		É

There is an inline shortcut to define the list entry data type constraints. See List Constraints .

For example:

ID	Name	Type	Constraint	Quality	É
0	MonthNames	list[string]	12[3]	N	É .

It is RECOMMENDED to put a maximum constraint on the list and list entry data types.

It is RECOMMENDED that a list entry data type be a struct, to enable the addition of new fields to the listÕs entries in the future.

- ¥ The cluster data version SHALL be incremented when the list order or entries change.
- ¥ An entry SHALL NOT be null.
- ¥ The list SHALL support reading and reporting all entries.
- ¥ The list SHALL support reporting, updates, and/or deletion of one or more entries.
- ¥ If the list is writable, it SHALL support writing or deleting the entire list.
- ¥ If the list is writable, it SHALL support updating one or more individual entries by indicating an index per updated entry.
- ¥ If the list is writable, it SHALL support deleting one or more individual entries by indicating an index per deleted entry.
- ¥ If the list is writable, it SHALL support adding one or more individual entries.
- ¥ A list MAY define an entry that is a struct that is fabric-scoped (see Fabric-Scoped Quality).

Fabric-Scoped List

- ¥ A fabric-scoped list SHALL define an entry data type that is a struct, which SHALL also be fab! ric-scoped (see Fabric-Scoped Struct).

Each entry in a fabric-scoped list SHALL be fabric-scoped to a particular fabric or no fabric.

A fabric-scoped list supports a fabric-filter that filters the view of the list for read and write interac! tions. This filter simplifies client side logic that does not want to read or write fabric data that is not associated with the accessing fabric .

- ¥ An interaction upon a list with fabric-filtering SHALL only indicate and access entries where the associated fabric matches the accessing fabric , and all other entries SHALL be ignored.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

¥ Fabric-filtered list entries SHALL be in the same order as the full list.

¥ Fabric-filtered list entries SHALL be indexed from 0 with no gaps, as if the other entries did not exist.

¥ For a write interaction, fabric-filtering SHALL be enabled.

¥ When writing to a fabric-scoped list, the write interaction SHALL be on an accessing fabric , oth! erwise, the write interaction SHALL fail (see Interaction Model).

¥ For a read interaction on a list, fabric-filtering MAY be enabled.

¥ For a read interaction on a list, with fabric-filtering disabled, the list SHALL be reported as a full list with all entries.

For example: A fabric-scoped full list with each entry having an associated FabricIndex and Value field:

```
list = [ { FabricIndex = A, Value = 20 },
Ê       { FabricIndex = B, Value = 30 },
Ê       { FabricIndex = A, Value = 40 },
Ê       { FabricIndex = B, Value = 50 },
Ê       { FabricIndex = B, Value = 60 } ]
```

would be a fabric-filtered list when accessed with fabric B:

```
list = [ { FabricIndex = B, Value = 30 },
Ê       { FabricIndex = B, Value = 50 },
Ê       { FabricIndex = B, Value = 60 } ]
```

Reading a fabric-filtered list entry index 2 accessed with fabric B reports:

```
list[2] = [ { FabricIndex = B, Value = 60 } ]
```

Writing fabric-filtered list entry index 1 when accessed with fabric B:

```
list[1] = [ { FabricIndex = B, Value = 55 } ]
```

changes the full list to:

```
list = [ { FabricIndex = A, Value = 20 },
Ê       { FabricIndex = B, Value = 30 },
Ê       { FabricIndex = A, Value = 40 },
Ê       { FabricIndex = B, Value = 55 },
Ê       { FabricIndex = B, Value = 60 } ]
```

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

7.18.1.10. Struct

A struct is a sequence of fields of any data type. Individual fields are identified by a field ID of unsigned integer, starting at 0 (zero), for the first field.

- A struct itself SHALL have no constraint qualities.
- Each struct field SHALL have its own qualities.
- Access, conformance and persistence qualities, when not not explicitly defined, SHALL be inher! ited from the instance of the struct itself.
- Struct fields MAY have optional conformance.
- A struct SHALL support reading and reporting of all fields.
- A struct SHALL support reporting changes to one or more fields.
- If the struct is writable, it SHALL support writing the entire struct.
- If a field of the struct is writable, the struct SHALL support updating the field.
- Because of optional struct field conformance, instances of the same struct MAY support multiple 'flavors' of the same struct data type, but with a different set of optional fields.

Fabric-Scoped Struct

- A fabric-scoped struct SHALL only be defined and occur as an entry in a fabric-scoped list.
- A fabric-scoped struct SHALL support the global FabricIndex field of type fabric-index , which indicates the associated fabric of the struct, or indicates that there is no associated fabric.
- The table that defines fields of a fabric-scoped struct SHALL NOT list the global FabricIndex field , which is a global field and defined implicitly.
- The global FabricIndex field of a fabric-scoped struct SHOULD NOT be indicated in a write inter! action.
- The global FabricIndex field of a fabric-scoped struct SHALL be ignored in a write interaction.
- When a write interaction creates a fabric-scoped struct entry (in a fabric-scoped list), the server SHALL implicitly load the accessing fabric-index into the global FabricIndex field of the struct.
- A fabric-scoped struct MAY be defined with some fields that are fabric-sensitive .
- For interactions on a fabric-scoped struct that report back data, fabric-sensitive struct fields SHALL NOT be indicated when reporting data back to the client, when the struct has an associ! ated fabric , and it is not the accessing fabric .

7.18.2. Derived Data Types

These data types are commonly used and derived from the base data types. If a data type is used by more than one cluster specification, then it SHALL be listed here as a derived data type. Such com! mon data types can then be reused instead of redefined in each cluster specification.

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Class	Data Type	Short	Base Type	ID	Size
Analog	Relative				
	Percentage units 1%	percent	uint8	0x32	1 bytes
	Percentage units 0.01%	percent100ths	uint16	0x33	2 bytes
	Time				
	Time of day	tod	struct	0xE0	4 bytes
	Date	date	struct	0xE1	4 bytes
	Epoch Time in Microseconds	epoch-us	uint64	0xE3	8 bytes
	Epoch Time in Seconds	epoch-s	uint32	0xE2	4 bytes
	UTC Time	utc	same as Epoch Time in Seconds but Deprecated		
	System Time in Microseconds	systime-us	uint64	0xE4	8 bytes

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Class	Data Type	Short	Base Type	ID	Size
Discrete	Enumeration				
	8-bit enumera! tion	enum8	uint8	0x30	1 byte
	16-bit enumer! ation	enum16	uint16	0x31	2 bytes
	Priority	priority	enum8	0x34	1 byte
	Status Code	status	enum8	0xE7	2 bytes
	Identifier				
	Fabric ID	fabric-id	uint64	0xD1	8 bytes
	Fabric Index	fabric-idx	uint8	0xD2	1 byte
	Node ID	node-id	uint64	0xF0	8 bytes
	IEEE Address	EUI64	same as Node ID but Deprecated		
	Group ID	group-id	uint16	0xF1	2 bytes
	Endpoint Num! ber	endpoint-no	uint16	0xE5	2 bytes
	Vendor ID	vendor-id	uint16	0xD3	2 bytes
	Device Type ID	devtype-id	uint32	0xED	4 bytes
	Cluster ID	cluster-id	uint32	0xE8	4 bytes
	Attribute ID	attrib-id	uint32	0xE9	4 bytes
	Field ID	field-id	uint32	0xEF	4 bytes
	Event ID	event-id	uint32	0xEE	4 bytes
	Command ID	command-id	uint32	0xEC	4 bytes
	Action ID	action-id	uint8	0xEA	1 bytes
	Transaction ID	trans-id	uint32	0xEB	4 bytes
	Index				
	Entry Index	entry-idx	uint16	0xF2	2 bytes
	Counter				
	Data Version	data-ver	uint32	0xD0	4 bytes
	Event Number	event-no	uint64	0xE6	8 bytes

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

Class	Data Type	Short	Base Type	ID	Size
Composite	String				
	Character String	string	octstr	0x42	desc
	Address				
	IP Address	ipadr	octstr	0xD3	4 or 16 bytes
	IPv4 Address	ipv4adr	octstr	0xD4	4 bytes
	IPv6 Address	ipv6adr	octstr	0xD5	16 bytes
	IPv6 Prefix	ipv6pre	octstr	0xD6	1 to 17 bytes
	Hardware Address	hwadr	octstr	0xD7	6 or 8 bytes

7.18.2.1. Time of Day

The Time of Day data type SHALL be a struct with these fields: Hours, Minutes, Seconds, and Hun! dredths.

The hours field represents hours according to a 24-hour clock. The range is from 0 to 23. The minutes field represents minutes of the current hour. The range is from 0 to 59. The seconds field represents seconds of the current minute. The range is from 0 to 59. The hundredths field repre! sents 100ths of the current second. The range is from 0 to 99. A value of null in any subfield indi! cates an unused subfield. If all subfields have a value of null, this indicates a null time of day.

7.18.2.2. Date

The Date data type SHALL be a struct with these fields:

The year - 1900 subfield has a range of 0 to 255, representing years from 1900 to 2155. The month field has a range of 1 to 12, representing January to December. The day of month field has a range of 1 to 31. Note that values in the range 29 to 31 may be invalid, depending on the month and year. The day of week field has a range of 1 to 7, representing Monday to Sunday. A value of null in any subfield indicates an unused subfield. If all subfields have a value of null, this indicates a null date.

7.18.2.3. Epoch Time in Microseconds

This type represents an offset, in microseconds, from 0 hours, 0 minutes, 0 seconds, on the 1st of January, 2000 UTC (the Epoch), encoded as an unsigned 64-bit scalar value.

This offset is the sum of two parts: time elapsed, not counting leap-seconds, and a local time offset. The local time offset MAY include a timezone offset and a MAY include a DST offset.

Any use of this type SHALL indicate how the associated local time offset is determined in the spe! cific context of that use. This MAY be done, for example, by simply saying the time is a UTC time, in which case the local time offset is 0.

A given Epoch Time value MAY be interpreted in at least two ways:

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

1. The value can be converted to a local clock date/time (year, month, day, hours, minutes, sec! onds, microseconds) by treating the local time offset as 0 and finding the UTC (year, month, day, hours, minutes, seconds, microseconds) tuple that corresponds to an elapsed time since the epoch time equal to the given value. The value then represents that tuple, but interpreted in the specific timezone and DST situation associated with the value. This procedure does not require knowing the local time offset of the value.
2. The value can be converted to a UTC time by subtracting the associated local time offset from the Epoch Time value and then treating the resulting value as an elapsed count of microseconds since the epoch time.

For example, an Epoch Time value of 0x0000_0BF1_B7E1_0000 corresponds to an offset of exactly 152 days. This can be interpreted as "00:00:00 on June 1, 2000" in whatever local time zone is associ! ated with the value. That corresponds to the following times in ISO 8601 notation:

- ¥ 2000-06-01T00:00Z if the associated local time offset is 0 (i.e. the value is in UTC).
- ¥ 2000-05-31T23:00Z if the associated local time offset is +1 hour (e.g. the CET timezone, without daylight savings).
- ¥ 2000-06-01T00:00+02 if the associated local time offset is +1 hour.
- ¥ 2000-06-01T04:00Z if the associated local time offset is -4 hours (e.g. the EDT time zone, which includes daylight savings).
- ¥ 2000-06-01T00:00-04 if the associated local time offset is -4 hours.

Conversion from NTP timestamps

Timestamps from NTP also do not count leap seconds, but have a different epoch. NTP 128-bit timestamps consist of a 64-bit seconds portion (NTP(s)) and a 64-bit fractional seconds portion (NTP(frac)). NTP(s) at 00:00:00 can be calculated from the Modified Julian Day (MJD) as follows:

NTP(s) = (MJD-15020) * (24*60*60)

where 15020 is the MJD on January 1, 1900 (the NTP epoch)

NTP(s) on January 1, 2000 00:00:00 UTC (MJD = 51544) is 3155673600 (0xBC17C200)

Epoch Time has a microsecond precision, and this precision can be achieved by using the most sig! nificant 32 bits of the fractional portion (NTP(frac32)). Conversion between the 128-bit NTP timestamps and a UTC Epoch Time in Microseconds is as follows:

UTC Epoch Time = (NTP(s) - 0xBC17C200)*10^6 + ((NTP(frac32)*10^6) / 2^32) where all numbers are treated as unsigned 64-bit integers and the division is integer division.

7.18.2.4. Epoch Time in Seconds

This type has the same semantics as Epoch Time in Microseconds , except that:

- ¥ the value encodes an offset in seconds, rather than microseconds;
- ¥ the value is encoded as an unsigned 32-bit scalar, rather than 64-bit.

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

This type is employed where compactness of representation is important and where the resolution of seconds is still satisfactory.

7.18.2.5. System Time in microseconds

System time in microseconds is an unsigned 64-bit value representing the number of microseconds since boot.

7.18.2.6. Percentage units 1%

A Percentage is an unsigned 8-bit value representing percent with a resolution of 1%. The range is from 0 (0%) to 100 (100%).

7.18.2.7. Percentage units 0.01%

A Percentage 100ths is an unsigned 16-bit value representing percent with a resolution of 0.01%. The range is from 0 (0.00%) to 10000 (100.00%).

7.18.2.8. Fabric-Index

This is an index that maps to a particular fabric on the node, see Fabric-Index . It is used for:

- ¥ the accessing fabric index of an interaction
- ¥ the FabricIndex global field in fabric-scoped data

7.18.2.9. Node ID

A 64-bit ID for a node scoped and unique to a particular fabric as indicated by an accompanying fabric-index adjacent instantiation.

7.18.2.10. Group ID

A 16-bit ID for a group scoped to a particular fabric as indicated by an accompanying fabric index adjacent instantiation.

7.18.2.11. Endpoint Number

An unsigned number that indicates an instance of a device type .

7.18.2.12. Vendor ID

A Vendor ID .

Vendor IDs MAY be used as a prefix in a Manufacturer Extensible Identifier format.

7.18.2.13. Device Type ID

An identifier that indicates conformance to a device type .

Device Type IDs SHALL be a Manufacturer Extensible Identifier . The specifics of its representation are described in Data Model Types .

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

7.18.2.14. Cluster ID

An identifier that indicates conformance to a cluster specification.

Cluster IDs SHALL be a Manufacturer Extensible Identifier . The specifics of its representation are described in Data Model Types .

7.18.2.15. Attribute ID

An identifier that indicates an attribute defined in a cluster specification.

Attribute IDs SHALL be a Manufacturer Extensible Identifier . The specifics of its representation are described in Data Model Types .

7.18.2.16. Field ID

An identifier that indicates a field defined in a struct.

Field IDs SHALL be a Manufacturer Extensible Identifier . The specifics of its representation are described in Data Model Types .

7.18.2.17. Event ID

An identifier that indicates an Event defined in a cluster specification.

Event IDs SHALL be a Manufacturer Extensible Identifier . The specifics of its representation are described in Data Model Types .

7.18.2.18. Command ID

An identifier that indicates a command defined in a cluster specification.

Command IDs SHALL be a Manufacturer Extensible Identifier . The specifics of its representation are described in Data Model Types .

7.18.2.19. Action ID

An identifier that indicates an action as defined in the Interaction Model specification.

7.18.2.20. Transaction ID

An identifier for a transaction as defined in the Interaction Model specification, see Transaction ID .

7.18.2.21. Entry Index

This is an index for a list data type .

7.18.2.22. Status Code

An enumeration value that means a success or error status. A status code is indicated as a response to an action in an interaction (see Interaction Model).

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

A status code SHALL be one of:

- ¥ a common status code from the set defined in the Interaction Model status code table .
- ¥ a cluster status code that is scoped to a particular cluster

The following table defines the the enumeration ranges for status codes.

Status Code Range	Description
0x00	common status code: SUCCESS
0x01	common status code: FAILURE
0x02 to 0x10	cluster scoped status codes
0x70 to 0xCF	other common status codes defined in Interaction Model Status Code Table .

Status codes in an undefined range, or status codes undefined within a range are reserved and SHALL NOT be indicated.

7.18.2.23. Priority

This is an enumeration of priority used to tag events and possibly other data. The data type does not define any particular ordering among the values. Specific uses of the data type may assign semantics to the values that imply an ordering relationship.

Value	Priority	Description
0	DEBUG	Information for engineering debugging/troubleshooting
1	INFO	Information that either drives customer facing features or provides insights into device functions that are used to drive analytics use-cases
2	CRITICAL	Information or notification that impacts safety, a critical function, or continued operation

7.18.2.24. Data Version

An unsigned number that indicates a Data Version .

7.18.2.25. Event Number

An unsigned number that indicates an Event instance.

7.18.2.26. Character String

The character string data type is derived from an octet string. The octets SHALL be characters with

Copyright © Connectivity Standards Alliance, Inc. All rights reserved.

UTF-8 encoding. An instance of this data type SHALL NOT contain truncated code points.

If at least one of the code points within the string has value 31 (0x1F), which is Unicode INFORMATION SEPARATOR 1 and ASCII Unit Separator , then any client making use of the string SHALL only consider the code points that appear before such an INFORMATION SEPARATOR 1 as being the textual informa! tion carried by the string. The remainder of the character string after a first INFORMATION SEPARATOR 1 is reserved for future use by this specification.

Note that the character string type is a bounded sequence of characters whose size bound format is not specified in the data model, but rather a property of the underlying encoding. Therefore, no assumptions are to be made about the presence or absence of a length prefix or null-terminator byte, or other implementation considerations.

It is RECOMMENDED to define constraints on the maximum possible string length.

7.18.2.27. IP Address

Either an IPv4 or an IPv6 address as defined below.

7.18.2.28. IPv4 Address

The IPv4 address data type is derived from an octet string. The octets SHALL correspond to the four octets in network byte order that comprise an IPv4 address represented utilizing quad-dotted nota! tion.

Examples of encoding:

- ¥ Address 192.168.2.235 * C0A802EB
- ¥ Address 10.4.200.75 * 0A04C84B

7.18.2.29. IPv6 Address

The IPv6 address data type is derived from an octet string. The octets SHALL correspond to the full 16 octets that comprise an IPv6 address as defined by RFCÊ4291The octets SHALL be presented in network byte order.

Examples of encoding:

- ¥ Address 2001:DB8:0:0:8:800:200C:417A* 20010DB80000000000080800200C417A
- ¥ Address 2001:0DB8:1122:3344:5566:7788:99AA:BBCC* 20010DB8112233445566778899AABBCC

7.18.2.30. IPv6 Prefix

The IPv6 prefix data type is derived from an octet string. The octets SHALL be encoded such that:

- ¥ The first octet SHALL encode the prefix length, in bits, in the range of 0 to 128.

  %A value of 0 indicates an absent/invalid prefix.

- ¥ The subsequent octets SHALL encode the contiguous leftmost bits of the prefix, in network byte order, with left justification, such that the first bit of the prefix is in the most significant bit of

 Copyright © Connectivity Standards Alliance, Inc. All rights reserved.