Data-Over-Cable Service Interface Specifications

DOCSIS® 4.0

Security Specification

CM-SP-SECv4.0-I04-220328

ISSUED

DISCLAIMER

This document is furnished on an "AS IS" basis and neither CableLabs nor its members provides any representation or warranty, express or implied, regarding the accuracy, completeness, noninfringement, or fitness for a particular purpose of this document, or any document referenced herein. Any use or reliance on the information or opinion in this document is at the risk of the user, and CableLabs and its members shall not be liable for any damage or injury incurred by any person arising out of the completeness, accuracy, or utility of any information or opinion contained in the document.

CableLabs reserves the right to revise this document for any reason including, but not limited to, changes in laws, regulations, or standards promulgated by various entities, technology advances, or changes in equipment design, manufacturing techniques, or operating procedures described, or referred to, herein.

This document is not to be construed to suggest that any company modify or change any of its products or procedures, nor does this document represent a commitment by CableLabs or any of its members to purchase any product whether or not it meets the characteristics described in the document. Unless granted in a separate written agreement from CableLabs, nothing contained herein shall be construed to confer any license or right to any intellectual property. This document is not to be construed as an endorsement of any product or company or as the adoption or promulgation of any guidelines, standards, or recommendations.

Document Status Sheet

Document Control Number	CM-SP-SECv4.0-I04-220328			
Document Title	Security Specification			
Revision History	D01 – 6/28/2019 I01 – 8/15/2019 I02 – 12/2/2020 I03 – 8/26/2021 I04 – 3/28/2022			
Date	March 28, 2022			
Status	~~Work in Progress~~	~~Draft~~	**Issued**	~~Closed~~
Distribution Restrictions	~~DOCSIS Focus Teams~~	~~CL/Member~~	~~CL/ Member/ Vendor~~	**Public**

Key to Document Status Codes

Work in Progress An incomplete document, designed to guide discussion and generate feedback that may include several alternative requirements for consideration.

Draft A document in specification format considered largely complete, but lacking review by Members and vendors. Drafts are susceptible to substantial change during the review process.

Issued A generally public document that has undergone Member and Technology Supplier review, cross-vendor interoperability, and is for Certification testing if applicable. Issued Specifications are subject to the Engineering Change Process.

Closed A static document, reviewed, tested, validated, and closed to further engineering change requests to the specification through CableLabs.

Trademarks

Contents

Figures

Tables

1 SCOPE

1.1 Introduction and Purpose

This specification is part of the DOCSIS family of specifications developed by Cable Television Laboratories (CableLabs). In particular, this specification is part of a series of specifications that define the sixth generation of high-speed data-over-cable systems, commonly referred to as the DOCSIS 4.0 specifications. This specification was developed for the benefit of the cable industry and includes contributions by operators and vendors from North and South America, Europe, and other regions.

1.2 Background

1.2.1 Broadband Access Network

A coaxial-based broadband access network is assumed. It may take the form of either an all-coax network or a hybrid fiber-coax network. The generic term "cable network" is used here to cover all cases.

A cable network uses a tree-and-branch architecture with analog transmission. The key functional characteristics assumed in this document are the following:

- Two-way transmission.
- A maximum optical/electrical spacing between the CMTS and the most distant CM of 100 miles (160 km) in each direction, although typical maximum separation may be 10-15 miles (16-24 km).

1.2.2 Network and System Architecture

1.2.2.1 The DOCSIS Network

The elements that participate in the provisioning of DOCSIS services are shown in Figure 1.

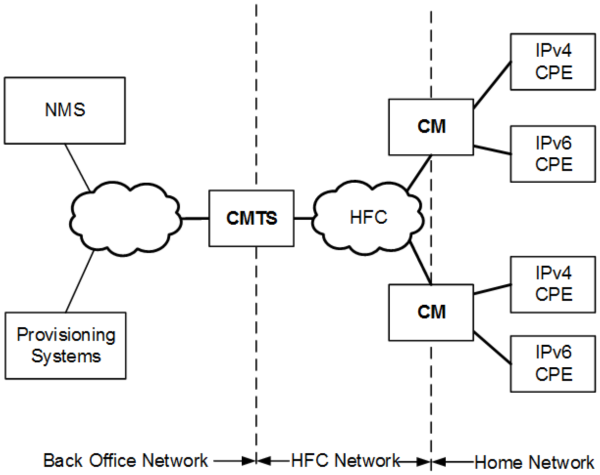

Figure 1 - The DOCSIS Network

The CM connects to the operator's cable network and to a home network, bridging packets between them. Many customer premises equipment (CPE) devices can connect to the CM's local area network (LAN) interfaces. CPE devices can be embedded with the CM in a single device, or they can be separate, standalone devices (as shown in Figure 1). CPE devices may use IPv4, IPv6, or both forms of IP addressing. Examples of typical CPE devices are home routers, set-top devices, and personal computers.

The CMTS connects the operator's back office and core network with the cable network. Its main function is to forward packets between these two domains and between the upstream and downstream channels on the cable network.

Various applications are used in the back office to provide configuration and other support to the devices on the DOCSIS network. These applications use IPv4 and/or IPv6, as appropriate to the particular operator's deployment. Some applications are listed below.

Provisioning Systems

- The DHCP servers provide the CM with initial configuration information, including IP address(es), when the CM boots.
- The Config File server is used to download configuration files to CMs when they boot. Configuration files are in binary format and permit the configuration of the CM's parameters.
- The Software Download server is used to download software upgrades to the CM.
- The Time Protocol server provides Time Protocol clients, typically CMs, with the current time of day.
- The Certificate Revocation server provides certificate status.

Network Management System (NMS)

- The SNMP Manager allows the operator to configure and monitor SNMP Agents, typically the CM and the CMTS.
- The Syslog server collects messages pertaining to the operation of devices.
- The IPDR Collector server allows the operator to collect bulk statistics in an efficient manner.

1.2.3 Service Goals

As cable operators have widely deployed high-speed data services on cable television systems, the demand for bandwidth has increased. Additionally, networks have scaled to such a degree that the space limitations of IPv4 addresses have become a constraint on network operations. To this end, CableLabs' member companies decided to add new features to the DOCSIS specification to increase channel capacity, enhance network security, expand addressability of network elements, and deploy new service offerings.

The DOCSIS system allows transparent bidirectional transfer of Internet Protocol (IP) traffic between the cable system headend and customer locations over an all-coax network or a hybrid fiber-coax (HFC) cable network. This network is shown in simplified form in Figure 2.

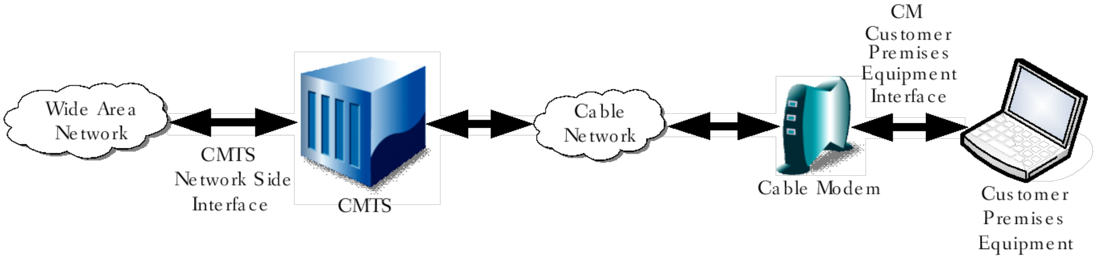
Wide Area Network
CMTS Network Side Interface
CMTS
Cable Network
CM Customer Premises Equipment Interface
Cable Modem
Customer Premises Equipment

Figure 2 - Transparent IP Traffic through the Data-Over-Cable System

1.2.4 Statement of Compatibility

This specification defines the DOCSIS 4.0 interface. Prior generations of DOCSIS technology were commonly referred to as the DOCSIS 1.0, 1.1, 2.0, 3.0, and 3.1 interfaces. DOCSIS 4.0 technology is backward-compatible with some equipment built to the previous specifications. DOCSIS 4.0-compliant CMs interoperate seamlessly with DOCSIS 3.1 and 3.0 CMTSs. DOCSIS 4.0-compliant CMTSs seamlessly support DOCSIS 3.1, 3.0, 2.0, and 1.1 CMs. Refer to Annex G of [DOCSIS MULPIv4.0] for general DOCSIS interoperability requirements.

Refer to Annex A for BPI+ compatibility requirements.

1.2.5 Reference Architecture

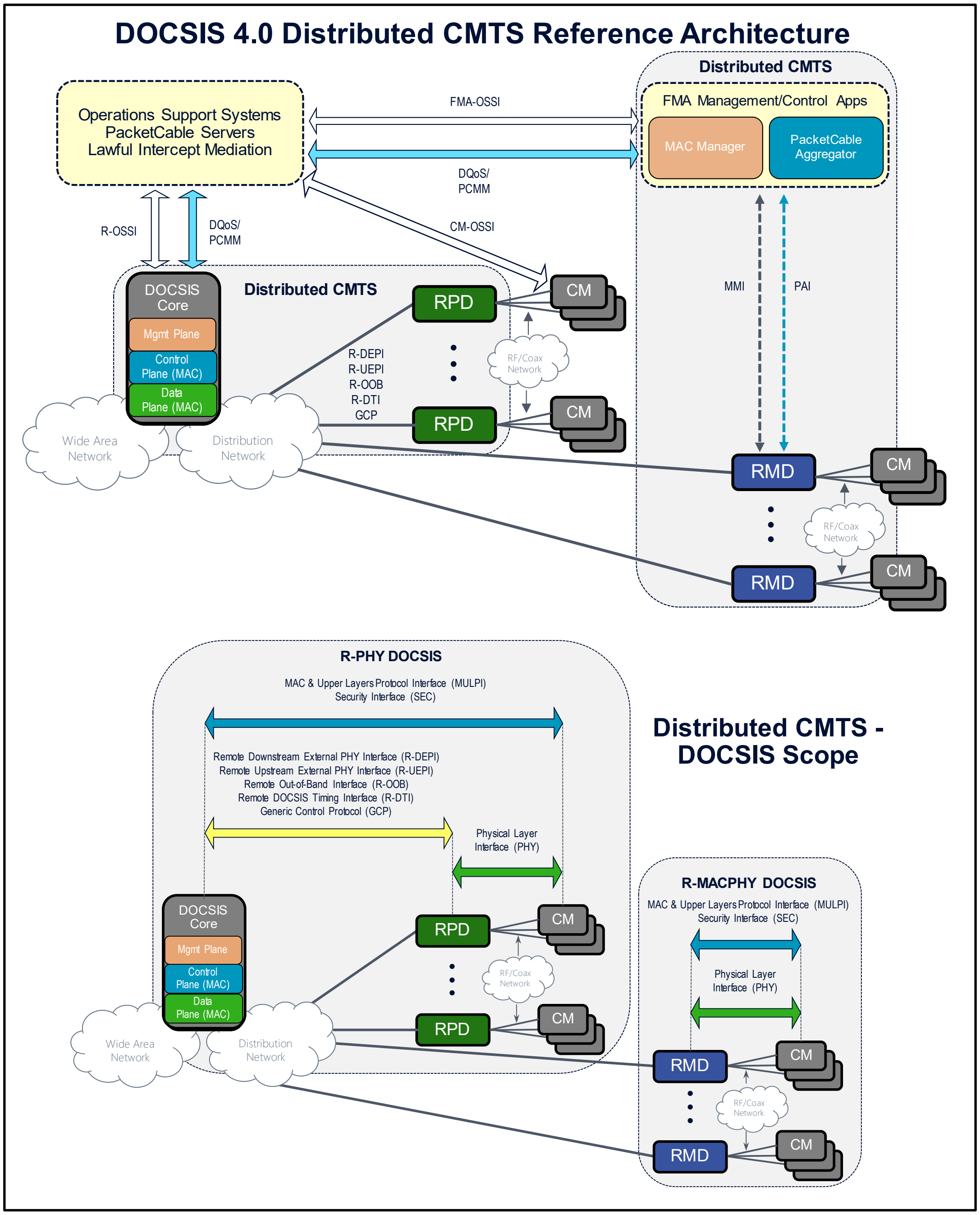

Figure 3 - Data-Over-Cable Reference Architecture

The reference architecture for data-over-cable services and interfaces is shown in Figure 3.

1.2.6 DOCSIS 4.0 Documents

A list of the specifications in the DOCSIS 4.0 series is provided in Table 1. For further information, please refer to https://www.cablelabs.com/specs.

Table 1 - DOCSIS 4.0 Series of Specifications

Designation	Title
CM-SP-PHYv4.0	Physical Layer Specification
CM-SP-MULPIv4.0	Media Access Control and Upper Layer Protocols Interface Specification
CM-SP-CM-OSSIv4.0	Cable Modem Operations Support System Interface Specification
CM-SP-CCAP-OSSIv4.0	Converged Cable Access Platform Operations Support System Interface Specification
CM-SP-SECv4.0	Security Specification
CM-SP-CMCIv3.0	Cable Modem CPE Interface Specification

This specification defines security requirements.

Related DOCSIS specifications are listed in Table 2.

Table 2 - DOCSIS 4.0 Related Specifications

Designation	Title
CM-SP-SECv3.1	Security Specification
CM-SP-CMCIv3.0	Cable Modem CPE Interface Specification
CM-SP-eDOCSIS	eDOCSIS™ Specification
CM-SP-DRFI	Downstream Radio Frequency Interface Specification
CM-SP-DTI	DOCSIS Timing Interface Specification
CM-SP-DEPI	Downstream External PHY Interface Specification
CM-SP-DSG	DOCSIS Set-Top Gateway Interface Specification
CM-SP-FMA-SYS	Flexible MAC Architecture System Specification
CM-SP-FMA-OSSI	Flexible MAC Architecture Operations Support System Interface Specification
CM-SP-FMA-MMI	Flexible MAC Architecture MAC Manager Interface Specification
CM-SP-FMA-PAI	Flexible MAC Architecture PacketCable Aggregator Interface Specification
CM-SP-GCP	Generic Control Plane Specification
CM-SP-L2VPN	Layer 2 Virtual Private Networks Specification
CM-SP-R-DEPI	Remote Downstream External PHY Interface Specification
CM-SP-R-DTI	Remote DOCSIS Timing Interface Specification
CM-SP-R-OOB	Remote Out-of-Band Specification
CM-SP-R-OSSI	Remote PHY OSS Interface Specification
CM-SP-R-PHY	Remote PHY Specification
CM-SP-R-UEPI	Remote Upstream External PHY Interface Specification
CM-SP-TEI	TDM Emulation Interfaces Specification

1.3 Requirements

Throughout this document, the words that are used to define the significance of particular requirements are capitalized. These words are:

"MUST" This word means that the item is an absolute requirement of this specification.

"MUST NOT" This phrase means that the item is an absolute prohibition of this specification.

"SHOULD" This word means that there may exist valid reasons in particular circumstances to ignore this item, but the full implications should be understood, and the case carefully weighed before choosing a different course.

"SHOULD NOT" This phrase means that there may exist valid reasons in particular circumstances when the listed behavior is acceptable or even useful, but the full implications should be understood, and the case carefully weighed before implementing any behavior described with this label.

"MAY" This word or the adjective "OPTIONAL" means that this item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because it enhances the product, for example; another vendor may omit the same item.

This document defines many features and parameters, and a valid range for each parameter is usually specified. Equipment (CM and CMTS) requirements are always explicitly stated. Equipment is to comply with all mandatory (MUST and MUST NOT) requirements to be considered compliant with this specification. Support of non-mandatory features and parameter values is optional.

1.4 Conventions

In this specification, the following convention applies any time a bit field is displayed in a figure. The bit field should be interpreted by reading the figure from left to right, then from top to bottom, with the MSB being the first bit so read and the LSB being the last bit so read.

```
MIB syntax and XML Schema syntax is represented by this code sample font.
```

Notices and/or Warnings are identified by this style font and label.

2 REFERENCES

2.1 Normative References

In order to claim compliance with this specification, it is necessary to conform to the following standards and other works as indicated, in addition to the other requirements of this specification. Intellectual property rights may be required to implement these references.

[C-DOCSIS] C-DOCSIS System Specification, CM-SP-CDOCSIS-I02-150305, March 5, 2015, Cable Television Laboratories, Inc.

[DOCSIS BPI+] Data-Over-Cable Service Interface Specifications, Baseline Privacy Plus Interface Specification, CM-SP-BPI+-C01-081104, November 4, 2008, Cable Television Laboratories, Inc.

[DOCSIS CCAP-OSSIv4.0] DOCSIS 4.0 CCAP Operations Support System Interface Specification, CM-SP-CCAP-OSSIv4.0-I06-220302, March 2, 2022, Cable Television Laboratories, Inc.

[DOCSIS CM-OSSIv4.0] DOCSIS 4.0 Cable Modem Operations Support System Interface Specification, CM-SP-CM-OSSIv4.0-I06-220302, March 2, 2022, Cable Television Laboratories, Inc.

[DOCSIS MULPIv4.0] DOCSIS 4.0 MAC and Upper Layer Protocols Interface Specification, CM-SP-MULPIv4.0-I05-220328, March 28, 2022, Cable Television Laboratories, Inc.

[DOCSIS RFIv2.0] DOCSIS 2.0 Radio Frequency Interface Specification, CM-SP-RFIv2.0-C02-090422, April 22, 2009, Cable Television Laboratories, Inc.

[DOCSIS SECv3.0] DOCSIS 3.0, Security Specification, CM-SP-SECv3.0-C01-171207, December 7, 2017, Cable Television Laboratories, Inc.

[DOCSIS SECv3.1] DOCSIS 3.1 Security Specification, CM-SP-SECv3.1-I10-211110, November 10, 2021, Cable Television Laboratories, Inc.

[FIPS 140-2] Federal Information Processing Standards Publication (FIPS PUB) 140-2, Security Requirements for Cryptographic Modules, June 2001.

[FIPS 180-4] Federal Information Processing Standards Publication (FIPS PUB) 180-2, Secure Hash Standard, May 2014.

[FIPS 197] Federal Information Processing Standards Publication (FIPS PUB) 197, Advanced Encryption Standard, November 2001.

[FIPS 46-3] Federal Information Processing Standards Publication (FIPS PUB) 46-3, Data Encryption Standard, October 1999.

[IEEE 802.1X] IEEE 802.1X-2010, IEEE Standard for Local and metropolitan area networks – Port-Based Network Access Control, February 2010.

[ISO 8859-1] ISO/IEC 8859-1:1998, 8-bit single-byte coded graphic character sets -- Part 1: Latin alphabet No.1.

[MMH] S. Halevi and H. Krawczyk, MMH: Software Message Authentication in Gbit/sec Rates, Proceedings of the 4th Workshop on Fast Software Encryption, (1997) vol. 1267 Springer-Verlag, pp. 172-189.

[NIST-800-38A] NIST-800-38A, Recommendation for Block Cipher Modes of Operation, Methods and Techniques, Morris Dworkin, 2001 Edition.

[PKCS#7] RSA Laboratories, PKCS #7: Cryptographic Message Syntax Standard, An RSA Laboratories Technical Note, Version 1.5, Revised November 1, 1993.

[PKT-SEC] PacketCable™ Security Specification, PKT-SP-SEC-C01-071129, November 29, 2007, Cable Television Laboratories, Inc.

[RFC 826] IETF RFC 826/STD0037, D.C. Plummer, Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48.bit Ethernet address for transmission on Ethernet hardware, November 1, 1982.

[RFC 1350] IETF RFC 1350/STD0033, The TFTP Protocol, (Revision 2), K. Sollins, July 1992.

[RFC 2104] IETF RFC 2104, HMAC: Keyed-Hashing for Message Authentication, H. Krawczyk et al., February 1997.

[RFC 2347] IETF RFC 2347, TFTP Option Extension, G. Malkin, A. Harkin, May 1998.

[RFC 2348] IETF RFC 2348, TFTP Blocksize Option, G. Malkin, A. Harkin, May 1998.

[RFC 2349] IETF RFC 2349, TFTP Timeout Interval and Transfer Size Options, G. Malkin, A. Harkin, May 1998.

[RFC 2616] IETF RFC 2616, Hypertext Transfer Protocol -- HTTP/1.1, R. Fielding, et al, June 1999

[RFC 3279] IETF RFC 3279, Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, W. Polk, et. al., April 2002.

[RFC 3376] IETF RFC 3376 Internet Group Management Protocol, Version 3, B. Cain, et al., October 2002.

[RFC 4131] IETF RFC 4131, Management Information Base for Data Over Cable Service Interface Specification (DOCSIS) Cable Modems and Cable Modem Termination Systems for Baseline Privacy Plus, S. Green et al., September 2005.

[RFC 4253] IETF RFC 4253, The Secure Shell (SSH) Transport Protocol Layer, T. Ylonen, C. Lonvick, January 2006.

[RFC 4861] IETF RFC 4861, Neighbor Discovery for IP version 6 (IPv6). T. Narten, E. Nordmark, W. Simpson, H. Soliman, September 2007.

[RFC 5216] IETF RFC 5216, The EAP-TLS Authentication Protocol, D. Simon, B. Aboba, R. Hurst, March 2008.

[RFC 5246] IETF RFC 5246, The Transport Layer Security (TLS) Protocol Version 1.2, T. Dierks, E. Rescorla, August 2008.

[RFC 5280] IETF RFC 5280, Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, W. Polk, May 2008.

[RFC 5281] IETF RFC 5281, Extensible Authentication Protocol Tunneled Transport Layer Security Authenticated Protocol Version 0 (EAP-TTLSv0), P. Funk, S. Blake-Wilson, August 2008.

[RFC 5652] IETF RFC 5652, Cryptographic Message Syntax (CMS), R. Housley, September 2009.

[RFC 5656] IETF RFC 5656, Elliptic Curve Algorithm Integration in the Secure Shell Transport Layer, D. Stebila, J. Green, December 2009.

[RFC 6960] IETF RFC 6960, X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, C. Adams, June 2013.

[RFC 6961] IETF RFC 6961, The Transport Layer Security (TLS) Multiple Certificate Status Request Extension, Y. Pettersen, June 2013.

[RFC 8446] IETF RFC 8446, The Transport Layer Security (TLS) Protocol Version 1.3, E. Rescorla, August 2018.

[RSA1] RSA Laboratories, PKCS #1: RSA Encryption Standard. Version 1.5, RSA Security, Inc., Bedford, MA, November 1993.

[RSA3] RSA Laboratories, PKCS #1 v2.0: RSA Cryptography Standard, October 1, 1999.

[SCTE 52] ANSI/SCTE 52 2018, Data Encryption Standard - Cipher Block Chaining Packet Encryption.

[X.509] ITU-T Recommendation X.509 (10/12): Information Technology - Open Systems Interconnection - The Directory: Public key and attribute certificate frameworks.

[X.690] ITU-T Recommendation X.690 (11/08) | ISO/IEC 8825-1:2002, Information Technology - ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).

2.2 Informative References

This specification uses the following informative references.

[DOCSIS CMCIv3.0] Data-Over-Cable Service Interface Specifications Cable Modem to Customer Premise Equipment Interface Specification, CM-SP-CMCIv3.0-I03-170510, May 10, 2017, Cable Television Laboratories, Inc.

[ISO 3166] ISO 3166-1, Codes for the representation of names of countries and their subdivisions -- Part 1: Country codes.

[NIST SP800-63B] NIST Special Publication 800-63B, Digital Identity Guidelines Authentication and Lifecycle Management, National Institute of Standards and Technology, June 2017.

[NIST SP800-90A] NIST Special Publication 800-90A, Recommendation for Random Number Generation Using Deterministic Random Bit Generators, Revision 1, National Institute of Standards and Technology, June 2015.

[RFC 2202] IETF RFC 2202, P. Cheng, R. Glenn, Test cases for HMAC-MD5 and HMAC-SHA-1, September 1997.

[RFC 3550] IETF RFC 3550/STD0064, H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, RTP: A Transport Protocol for Real-Time Applications, July 2003.

[RFC 4086] IETF RFC 4086, D. Eastlake, J. Schiller, S. Crocker, Randomness Requirements for Security, June 2005.

[RSA2] RSA Laboratories, Some Examples of the PKCS Standards, RSA Data Security, Inc., Bedford, MA, November 1, 1993.

[SET Book 2] SET, Secure Electronic Transaction Specification Book 2: Programmer's Guide, Version 1.0, May 31, 1997.

2.3 Reference Acquisition

- Cable Television Laboratories, Inc., 858 Coal Creek Circle, Louisville, CO 80027; Phone +1-303-661-9100; Fax +1-303-661-9199. http://www.cablemodem.com.
- Federal Information Processing Standards: 100 Bureau Drive, Mail Stop 3200, Gaithersburg, MD 20899-3200. Phone +1-301-975-4054; Fax +1-301-926-8091. http://csrc.nist.gov/publications/fips/.
- IETF Secretariat, c/o Corporation for National Research Initiatives, 1895 Preston White Drive, Suite 100, Reston, VA 20191-5434 Phone +1-703-620-8990; Fax +1-703-620-9071. http://www.ietf.org.
- ITU Recommendations: Place des Nations, CH-1211, Geneva 20, Switzerland. Phone +41-22-730-51-11; Fax +41-22-733-7256. http://www.itu.int.
- Public Key Cryptography Standards: RSA Security Inc. 174 Middlesex Turnpike, Bedford, MA 01730. Phone +1-781-515-5000; Fax 781-515-5010. http://www.rsasecurity.com/rsalabs/.
- SCTE, Society of Cable Telecommunications Engineers, 140 Philips Road, Exton, PA 19341-1318, Phone +1-800-542-5040; Fax+1-610-363-5898, http://www.scte.org/default.aspx/.

3 TERMS AND DEFINITIONS

This specification uses the following terms.

Bridging CMTS A CMTS that makes traffic forwarding decisions between its Network System Interfaces and MAC Domain Interfaces based upon the Layer 2 Ethernet MAC address of a data frame.

Difficulty Level of effort significantly above the typical skills and capabilities of an ordinary practitioner of the relevant technical area, such that most practitioners would not be capable of performing the task.

DER Encoded A value which is encoded using the ASN.1 Distinguished Encoding Rules [X.690].

Downstream Flow of signals from the cable system control center through the distribution network to the customer. For communication purposes, associated with transmission (down) to the end user.

Dynamically joined Multicast Sessions Multicast sessions joined after cable modem registration.

Hardware Includes software and CPU and instructions and data that are permanently embedded in such device or component in a form that cannot be modified or updated using *widely available tools* and can only be modified or updated using *professional tools* with *difficulty*.

Key transition period The time period in which an Authentication Key that is near its expiration is replaced by a new Authentication Key through a negotiated update process between the CMTS and the CM.

Legacy PKI The legacy PKI is defined by the DOCSIS 3.0 Security specification [DOCSIS SECv3.0] and supports backward compatibility with devices implementing older versions of DOCSIS Security; the term "legacy" can refer to aspects of the legacy PKI such as legacy certificates and legacy keys.

MAC domain A logical link layer network consisting of a common address scheme (such as IEEE 802.3 Ethernet) in which elements may send and receive OSI Layer 2 messages between and among one another. MAC domain boundaries may be established through both physical and logical means; separate channels or subchannels utilizing differing frequency and/or encoding methods or assigning separate bundles/bridge groups or subinterfaces to common frequency-domain channels or subchannels.

Professional Tools Instruments that require specialized skills and are only usable by an expert with specialized knowledge of system-on-a-chip technology, such as, but not limited to, lasers, in-circuit emulators, focused ion beam devices, electron microscopes, x-ray or other optical inspection instruments, logic analyzers with 300-MHz or greater input bandwidth, or chip disassembly systems.

Routing CMTS A CMTS that makes traffic forwarding decisions between its Network System Interfaces and MAC Domain Interfaces based upon the Layer 3 (network) address of a packet.

Software An implementation that includes but is not limited to DOCSIS 4.0 functions through a CPU executing computer program code consisting of instructions or data, other than such instructions or data that are included in *hardware*, where such instructions or data can be modified by download or by any manner of update.

Hardware is a physical device, including a component that implements any part of the DOCSIS 4.0 requirements.

Static Multicast Sessions Multicast sessions joined during cable modem registration.

Trust Anchor An authoritative entity for which trust is assumed and not derived. In DOCSIS 4.0, the root certificate acts as the trust anchor from which the chain of trust is derived.

Upstream Traffic and paths that go from the subscriber to the headend.

Widely Available Tools Instruments that are generally available at a reasonable price and that do not require specialized knowledge of system-on-a-chip technology to operate, such as, but not limited to, screwdrivers, jumper clips, discrete logic components, soldering irons, and specialized electronic or software tools such as memory readers and writers, logic analyzers with less than 300-MHz input bandwidth, debuggers, de-compilers or similar software development operations, or emulators.

4 ABBREVIATIONS AND ACRONYMS

This specification uses the following abbreviations and acronyms.

3DES	Triple encryption with the Data Encryption Standard
AES	Advanced Encryption Standard
AK	Authorization Key
ARP	Address Resolution Protocol
ASN.1	Abstract Syntax Notation 1
BCP	Best Current Practice
BPI+	Baseline Privacy Interface Plus
BPKM	Baseline Privacy Key Management
CA	Certificate Authority
CBC	Cipher Block Chaining
CCAP	Converged Cable Access Platform
CDS	Credential Data Structure
CFB	Cipher Feedback
CIDR	Classless Inter-Domain Routing
CM	Cable Modem
CMCI	Cable Modem to Customer Premises Equipment Interface
CMS	Cryptographic Message Structure
CMTS	Cable Modem Termination System
CPE	Consumer Premises Equipment
CRC	Cyclic Redundancy Check
CRL	Certificate Revocation List
CTR	Counter: the counter mode of a block cipher
CVC	Code Verification Certificate
CVS	Code Verification Signature
DA	Destination Address
DAA	Distributed Access Architecture
DER	Distinguished Encoding Rules
DES	Data Encryption Standard
DHCP	Dynamic Host Configuration Protocol
DHCPv4	Version of DHCP for IPv4
DHCPv6	Version of DHCP for IPv6
DOCSIS	Data-Over-Cable Service Interface Specifications
DoS	Denial of Service
DSID	Downstream Service Identifier

EAE	Early Authentication and Encryption
ECB	Electronic Code Book
ECDSA	Elliptic Curve Digital Signature Algorithm
EDE	Encrypt-Decrypt-Encrypt
EH	Extended Header
EHDR	Extended MAC Header
EKU	Extended Key Usage
Fn	The nth Fermat number
FC	Frame Control
FCRC	Fragment Cyclic Redundancy Check
FIPS	Federal Information Processing Standards
FQDN	Fully Qualified Domain Name
FSM	Finite State Machine
FWAH	Firmware Authentication Header
HCS	Header Check Sequence
HFC	Hybrid Fiber/Coax
HMAC	Keyed-Hash Message Authentication Code
IEEE	Institute of Electrical and Electronics Engineers
IGMP	Internet Group Management Protocol
IP	Internet Protocol
IPR	Intellectual Property Rights
IPv4	Version 4 of the Internet Protocol
IPv6	Version 6 of the Internet Protocol
ISO	International Organization for Standards
ITU-T	Telecommunication Standardization Sector of the International Telecommunications Union
IV	Initialization Vector
KEK	Key Encryption Key
LAN	Local Area Network
LSB	Least Significant Bit
MAC	Media Access Control
MDD	MAC Domain Descriptor
MGF	Mask Generation Function
MIB	Management Information Base
MIC	Message Integrity Check
MLD	Multicast Listener Discovery
MMH	Multilinear Modular Hash
MMM	MAC Management Message

MSB	Most Significant Bit
MSO	Multiple Systems Operator
NMS	Network Management System
NFV	Network Functions Virtualization
OCSP	Online Certificate Status Protocol
OID	Object Identifier
OSS	Operations Support System
OUI	Organizationally Unique Identifier
PDU	Protocol Data Unit
PFS	Perfect Forward Secrecy
PHS	Payload Header Suppression
PKI	Public Key Infrastructure
PSA	Persistent Security Attribute
QoS	Quality of Service
RF	Radio Frequency
RFC	Request For Comments
RSA	Rivest, Shamir, Adleman (a public key cryptographic algorithm)
RTP	Real-time Transport Protocol
SA	Security Association
SA	Source Address
SAID	Security Association Identifier
SAV	Source Address Verification
SCCA	SSH Client Credential Authentication
SET	Secure Electronic Transaction
SHA-1	Secure Hash Algorithm 1
SID	Service Identifier
SNMP	Simple Network Management Protocol
SSD	Secure Software Download
SSH	Secure Shell
TEK	Traffic Encryption Key
TFTP	Trivial File Transfer Protocol
TLS	Transport Layer Security
TLV	Type/Length/Value
ToD	Time of Day
TOFU	Trust On First Use
UTC	Coordinated Universal Time
XOR	Exclusive Or

5 OVERVIEW

The intent of this specification is to describe security services for DOCSIS 4.0 communications. It has two main goals:

1. to provide cable modem users with data privacy across the cable network and
2. to prevent unauthorized users from gaining access to the network's radio frequency (RF) MAC services.

This specification provides operators with tools to secure the provisioning process of cable modems (CMs) and protect CM users by encrypting traffic flows between the CM and the cable modem termination system (CMTS).

The protected RF MAC data communications services fall into three categories:

1. best-effort high-speed IP data services,
2. data services with guaranteed quality of service (QoS), and
3. IP multicast group services.

The CMTS protects against unauthorized access to these data transport services by enforcing encryption of the associated traffic flows across the cable network. DOCSIS technology employs an authenticated client/server key management protocol in which the CMTS (the server) controls distribution of keying material to CMs (the clients).

The system defined by this specification is used to protect packets on the cable network and is based on DOCSIS Baseline Privacy Plus [DOCSIS BPI+].

This specification explicitly assumes the following to be true:

- It is acceptable that DOCSIS 4.0 security features are designed to provide a reasonable level of security for a 10-year time frame.
- The operational life of a CM will not exceed 20 years.
- The interface between the CMTS and provisioning servers is secure.
- CM provisioning servers are trusted.
- At the beginning of the provisioning process, the CM is not trusted. The CM increases its level of trust as it successfully completes different provisioning steps. These steps include device authentication, registration request validation, and service authorization. Once the CM successfully completes the provisioning process with DOCSIS 4.0 security features enabled, it is considered trusted enough to provide data services to subscribers.
- Threat to integrity of encrypted data on the cable network link is low.
- To be backward compatible with existing devices, it is acceptable to continue the support of features that are considered to have low-level security characteristics. MSOs can enable/disable these features as needed.

5.1 DOCSIS 4.0 Security Features

The DOCSIS 4.0 specification defines a certificate public key infrastructure (PKI) that strengthens the security of CM authentication and secures software download features.

This specification defines the Baseline Privacy Plus Interface (BPI+) architecture, which covers CM authentication, key exchange, and establishing encrypted traffic sessions between the CM and CMTS. Early Authentication and Encryption (EAE) applies BPI+ earlier in the provisioning process (see Section 8). This specification also defines security features for the CM provisioning process, which includes Secure Software Download (SSD).

This specification introduces versioning to the BPI+ feature. Previous to this specification, there was just BPI+, which in this specification is now referred to as BPI+ Version 1, or just BPI+ V1. New to this specification is BPI+ Version 2, or just BPI+ V2. BPI+ V2 introduces new features including message authentication, network authentication (i.e., mutual authentication), and perfect forward secrecy. The term BPI+ is used when there is a statement which applies to any version of BPI+.

5.1.1 BPI+ Architecture

BPI+ has two component protocols.

- An encapsulation protocol for encrypting packet data across the cable network, defining
  1. the frame format for carrying encrypted packet data within DOCSIS MAC frames;
  2. a set of supported cryptographic suites, i.e., pairings of data encryption and authentication algorithms; and
  3. the rules for applying those algorithms to a DOCSIS MAC frame's packet data.
- A key management protocol (Baseline Privacy Key Management, BPKM) for providing secure distribution of keying data from CMTSs to CMs. Through BPKM, the CM and CMTS synchronize keying data; in addition, the CMTS uses the protocol to implement conditional access to network services.

5.1.1.1 Packet Data Encryption

DOCSIS encryption services are defined as a set of extended services within the DOCSIS MAC sublayer. Packet Header information specific to security is placed in a Baseline Privacy Extended Header (EH) element within the MAC EH (see [DOCSIS MULPIv4.0]).

DOCSIS technology encrypts only the MAC frame's packet data; the header of the DOCSIS MAC frame is never encrypted. DOCSIS MAC management messages, except REG-REQ-MP messages, are always unencrypted. REG-REQ-MP messages are encrypted when EAE is enabled. Section 6 specifies the format of DOCSIS MAC frames carrying encrypted packet data payloads.

5.1.1.2 Key Management Protocol

CMs use the BPKM protocol (see Section 7) to obtain authorization and traffic encryption keying material from the CMTS and to support periodic reauthorization and key refresh. The BPKM protocol uses digital certificates, a public-key encryption algorithm, and two-key, 3DES to secure key exchanges between the CM and the CMTS.

The BPKM protocol adheres to a client/server model in which the CM, a BPKM client, requests keying material and the CMTS, a BPKM server, responds to those requests, ensuring that individual CM clients receive only keying material for which they are authorized. The BPKM protocol is transported over DOCSIS MAC management messages.

DOCSIS technology uses public-key cryptography to establish a shared secret (an Authorization Key) between the CM and its CMTS. The shared secret is then used to derive secondary keys, which are in turn used to secure subsequent BPKM exchanges of traffic encryption keys. This two-tiered mechanism for key distribution permits traffic encryption keys to be updated without incurring the overhead of computationally intensive public-key operations.

A CMTS authenticates a client CM during the initial authorization exchange, which occurs when EAE is enabled or when post-registration BPI+ is enabled (see Section 8). Each CM carries a unique digital certificate issued by the CM's manufacturer. The digital certificate contains the CM's public key along with the CM MAC address, the identity of the manufacturer, and the CM serial number. When requesting an Authorization Key, a CM presents its digital certificate to a CMTS. The CMTS verifies the digital certificate, then uses the CM's public key to encrypt an Authorization Key, which the CMTS sends to the requesting CM.

5.1.1.3 DOCSIS Security Associations

A DOCSIS Security Association (SA) is the set of security information shared by a CMTS and one or more of its client CMs in order to support secure communications across the cable network.

There are three types of DOCSIS SAs: Primary, Static, and Dynamic. A Primary SA is tied to a single CM and is established when that CM completes authentication. Static SAs can be shared by multiple CMs and are established, based on a CMTS configuration, when a CM completes authentication. Dynamic SAs can be shared by multiple CMs and are normally established dynamically in response to the request of initiation of specific downstream traffic flows.

- An SA's shared information comprises the cryptographic suite, traffic encryption keys and cipher block chaining (CBC) initialization vectors, and the lifetime of the keying information. Each SA is identified with a 14-bit handle, known as a Security Association Identifier (SAID).
- Each CM on which security is enabled establishes a Primary SA with its CMTS. When the CM encrypts upstream traffic, including the REG-REQ-MP MAC management message, it MUST use the CM's Primary SA. The value of the Primary SAID is established during the initial authorization exchange.

- Downstream traffic may be encrypted under any of the three types of SAs. A downstream IP multicast data packet, for example, is typically intended for multiple CMs and therefore is usually encrypted under a Static or Dynamic SA. Downstream unicast traffic directed at CPE devices behind the CM are typically encrypted under the CM's Primary SA.

A CM MUST support:

- A Primary SA;
- A minimum of 15 SAs, each of which can be used as either a Dynamic SA or Static SA.

A CMTS MUST support:

- A Primary SA for every CM;
- At least one Dynamic SA (per CMTS).

A CMTS MAY support Static SAs.

Using the BPKM protocol, a CM requests from its CMTS and SA's keying material. The CMTS ensures that each client CM accesses only those SAs that it is entitled to access.

An SA's keying material (e.g., the key and CBC initialization vector) has a limited lifetime. When the CMTS delivers SA keying material to a CM, it also provides the CM with that material's remaining lifetime. It is the responsibility of the CM to request new keying material before the current keying material expires. The BPKM protocol specifies how the CM and CMTS maintain key synchronization.

5.1.1.4 QoS SIDs and DOCSIS SAIDs

The Baseline Privacy EH element in downstream DOCSIS MAC frames (see [DOCSIS MULPIv4.0]) contains the DOCSIS SAID under which the downstream frame is encrypted. If the downstream frame is a unicast packet addressed to a CPE device behind a particular CM, then the frame will typically be encrypted under the CM's Primary SA. If the downstream frame is a multicast packet intended for receipt by multiple CMs, then the extended header element will contain the Static or Dynamic SAID mapped to that multicast group. The SAID (Primary, Static or Dynamic), in combination with other data fields in the downstream extended header element, identifies to a receiving modem the particular set of keying material required to decrypt the DOCSIS MAC frame's encrypted Packet Data field. See Section 6 for details of the MAC frame format.

Because all upstream traffic is encrypted under the CM's Primary SA, upstream DOCSIS MAC frames do not carry an SAID in their extended headers; instead, the Baseline Privacy EH element may contain a valid QoS Service Identifier (SID) assigned to the CM (see Section 6.3 for details).

The Baseline Privacy EH element serves multiple purposes in upstream DOCSIS PDU MAC frames. As an alternative to identifying the particular set of keying material used to encrypt a frame's packet data, certain Baseline Privacy EH elements provide a mechanism for issuing piggybacked bandwidth requests (see Section 6.3.1); in some cases, it can also carry fragmentation control data (see Section 6.4). These two functions are tied to a particular QoS SID; for this reason, the relevant upstream Baseline Privacy EH elements contain a QoS SID rather than a Primary SAID. The SAID can be deduced by the CMTS from the QoS SID and the logical upstream on which the MAC frame was received.

The SAID associated with a Primary, Static, or Dynamic SA can be any 14-bit value. A CMTS MUST NOT assign the same SAID to more than one type of SA within a MAC Domain Downstream Service Group (MD-DS-SG). In other words, if an SAID is being used as a Primary SA, then the CMTS cannot use the same SAID for a Static or Dynamic SA within the same MD-DS-SG. Likewise, if a CMTS has assigned an SAID to a Static or Dynamic SA, then that value cannot be used as the Primary SAID for any modem within that MD-DS-SG. Additionally, a CMTS MUST NOT assign the same SAID to more than one non-Primary (Static or Dynamic) SA within an MD-DS-SG.

5.1.1.5 BPI+ Enforce

BPI+ authentication includes digital certificate CM MAC address validation, which helps prevent CM MAC address cloning and theft of service. Hackers could modify a CM's behavior in order to bypass BPI+ authentication and steal service using MAC address cloning. Enforcing BPI+ on the CMTS will help prevent this type of attack. The CMTS MUST support the following configurable BPI+ Enforce policies for each MAC domain:

Policy 0: Disable. The CMTS does not enforce BPI+.

Policy 1: 1.1 Style Config File Parameters and Capability. The CMTS enforces BPI+ on CMs that register with BPI+ enabled (missing TLV 29 or containing TLV 29 set to enable) and with a Modem Capabilities Privacy Support TLV (5.6) set to BPI+ support.

Policy 2: 1.1 Style Config File Parameters. The CMTS enforces BPI+ on CMs that register with parameters indicating BPI+ is enabled (missing TLV 29 or containing TLV 29 set to enable).

Policy 3: 1.1 Style Config File. The CMTS enforces BPI+ on CMs that register with a DOCSIS 1.1 style configuration file. Because DOCSIS 4.0 devices will only register with a DOCSIS 1.1 style configuration file, Policy 3 is the same as Policy 4 when using DOCSIS 4.0 technology.

Policy 4: Total. The CMTS enforces BPI+ on all CMs.

Policies 1, 2, and 3 support a mixed network of DOCSIS 1.1 and higher CMs. Policy 4 is the most effective configuration for preventing CM MAC address cloning.

When enforcing BPI+ on a CM, a CMTS MUST NOT forward post-registration traffic in either direction for that CM until it has successfully established a Primary SA using EAE or post-registration BPI+ provisioning. BPI+ Enforce does not conflict with EAE. A DOCSIS 3.0 or higher CM that successfully performs EAE before registration will meet the BPI+ Enforce requirements that are applied after registration.

The CMTS MUST support the capability to exclude individual CMs from BPI+ Enforce based on their MAC addresses on a per-MAC domain basis. If a CM is on the exclusion list, then the CMTS MUST NOT enforce BPI+ on that CM. This means that the CMTS will support BPI+ if initiated by that CM, but it will not enforce it.

5.1.2 Secure Provisioning

The processes used to provision a CM are DHCP, time of day (ToD), and TFTP at the IP layer and registration at the MAC layer. Securing these provisioning processes plays a critical role in protecting the CMs and the network from attacks and in preventing service theft. The CMTS can help secure these processes by assuring that traffic is only forwarded to/from CMs and CPEs, with source IP addresses that have been assigned by the MSO. The CMTS can also verify that a CM is registering with the correct service parameters by learning CM provisioning information in DHCP and TFTP messaging flows.

Securing the CM software image download process helps assure that the CM is running the correct/authorized version of code. Authenticating the source and verifying the integrity of downloaded code is vital to the overall operation and security of DOCSIS-based networks. When properly triggered, the CM downloads a software image from a Software Download server and validates the new software image's digital signature before installation.

5.2 Operation

5.2.1 Cable Modem Initialization

The [DOCSIS MULPIv4.0] specification divides CM initialization into the following sequence of tasks:

- scan for downstream channel and establish synchronization with the CMTS,
- obtain transmit parameters,
- perform ranging,
- establish IP connectivity,
- establish time of day,
- transfer operational parameters (download configuration file), and

- CMTS registration.

BPI+ security may be established between the CM and the CMTS following CM ranging (EAE) or following registration. Ordinarily, BPI+ is established using EAE immediately following ranging and remains in effect at least until registration is complete. Parameters within the configuration file instruct the CM whether to maintain BPI+ security with the CMTS or to operate in an unsecured mode from that point onward. The CM runs with BPI+ security enabled unless BPI+ is explicitly disabled in the DOCSIS configuration file.

BPI+ security initialization begins with the CM sending the CMTS an Authentication Information message containing the CM's Device Certificate Authority (CA) Certificate. The Authorization Request message contains fields identifying CM, the CM's cipher capabilities, information for establishing an authorization key, and message integrity protection.

If the CMTS successfully authenticates the requesting CM, it responds with an Authorization Reply containing the Primary SAID for the CM and fields for establishing the Authorization Key. The CM decrypts the encrypted Authorization Key using the private key of its certificate. From the Authorization Key, the CM and the CMTS derive the keys needed to secure a CM's subsequent requests for traffic encryption keys and the CMTS's responses to these requests.

The Authorization Reply also contains a list of SA descriptors, which identify the Primary and Static SAs that the requesting CM is authorized to access. Each SA descriptor consists of a collection of SA parameters, including the SA's SAID, type, and cryptographic suite. The list contains at least one entry: a descriptor describing the CM's Primary SA. Additional entries are optional and describe any Static SAs that the CM is permitted to access.

After successfully completing authentication and authorization with the CMTS, the CM sends key requests to the CMTS, requesting traffic encryption keys for each of the SA descriptors it received in the Authorization Reply. The CM's traffic key requests are authenticated using a keyed hash (the HMAC algorithm [RFC 2104]); the Message Authentication Key is derived from the Authorization Key obtained during the authorization exchange. The CMTS responds to each key request with a Key Reply message containing Traffic Encryption Keys (TEKs); TEKs are 3DES encrypted with a Key Encryption Key (KEK) derived from the Authorization Key. Like the Key Requests, Key Replies are authenticated with a keyed hash where the Message Authentication Key is derived from the Authorization Key.

BPI+ security initialization ends when the CM has received the Key Reply messages associated with all the SAIDs identified in the Authorization Reply message except those with cryptographic suites not supported by the CM.

5.2.1.1 Network Admission Control

Restricting network access to only authorized devices is an important part of maintaining security. One of the methods used to provide this function is CM authentication using [X.509] certificates. Certificates are issued to trusted vendor companies for installation in their CMs. When EAE or post-registration BPI+ is enabled, each CM is authenticated by the CMTS, using these certificates, before completing the provisioning process. If the CMTS is not able to validate the CM, then it does not allow the CM to access the network.

Forcing all CMs to successfully complete EAE or post-registration BPI+ before allowing them access to the network can help prevent hackers from using older CMs to bypass authentication. This capability for EAE BPI+ is defined in this specification (see Section 8), and for post registration, BPI+ is defined in the [DOCSIS MULPIv4.0] specification.

5.2.1.2 EAE and Authentication Reuse

When EAE is enabled, CM authentication normally occurs following ranging completion and is in turn followed by key exchanges and encrypted session establishment (see Section 8). Because of this, the subsequent provisioning messages (DHCP, ToD, and TFTP) are secured between the CM and CMTS. This step allows these provisioning applications to reuse the authentication and encryption functions of the CM for secure communication and avoids having to set up a separate secure session for each application, which would result in longer provisioning times and would restrict other security features such as configuration file learning (see Section 9.4.2.4).

5.2.1.3 Configuration Registration Enforcement

To help prevent theft-of-service attacks caused by alterations to the configuration file, the CMTS TFTP proxy requirements are specified herein. These requirements enable the CMTS to hide the true address of the TFTP server from the CM and other devices on the cable network and to learn the contents of the configuration file independently of the configuration information that is sent to it by the CM. When this feature is enabled, the CMTS proxies the TFTP messages between the CM and TFTP server. The CM is provisioned with the CMTS IP address instead of the actual address of the configuration file server, helping hide the TFTP server address from exposure on the cable network, where it might be captured and used for a denial of service (DoS) attack.

The CMTS may also learn the contents of the configuration file as it relays DHCP messages and proxies the information between the configuration file server and the CM. With this information, the CMTS can verify that the CM is downloading the correct configuration file and that the registration request contains the correct CM configuration settings.

5.2.2 Cable Modem Key Update Mechanism

The TEKs that the CMTS provides to client CMs have limited lifetimes. The CMTS delivers a key's remaining lifetime, along with the key value, in the Key Reply messages that it sends to its client CMs. The CMTS controls which keys are current by flushing expired keys and generating new keys as required. It is the responsibility of individual CMs to ensure that the keys they are using match those the CMTS is using. CMs do this by tracking when a particular SAID's key is scheduled to expire and issuing a new Key Request prior to that time.

In addition, CMs are required to periodically reauthorize with the CMTS (see Section 7.1.6). As for TEKs, an Authorization Key has a finite lifetime that the CMTS provides to the CM along with the key value. It is the responsibility of each CM to reauthorize and obtain a fresh Authorization Key (and an up-to-date list of SA descriptors) before the CM's current Authorization Key expires. This intermediate period just before the expiration of the Authorization Key is called the key transition period.

5.2.3 Cable Modem Secure Software Download

To download a CM software image securely, the CM vendor and/or MSO will digitally sign the image using the appropriate code verification certificate (CVC) and place the image on a Software Download server. A CM is enabled to download a software image when it receives a valid CVC in its configuration file. Triggering the download of the software image can be done using parameters in the CM configuration file or using SNMP commands.

After a CM downloads a software image, it validates the image by verifying that the included CVC chains to the Root CA Certificate trust anchor and by checking the image's digital signature. If this validation is successful, then it installs the software image for operation.

6 ENCRYPTED DOCSIS MAC FRAME FORMATS

6.1 CM Requirements

When operating with BPI+ security enabled, the CM MUST encrypt the Protocol Data Unit (PDU) regions of all the following types of DOCSIS MAC frames transmitted on to the cable network:

- Variable-length PDU MAC Frames;
- Fragmentation MAC Frames;
- Registration Request (REG-REQ-MP) MAC Management Message Frames.

In each of these cases, a Baseline Privacy EH element in the DOCSIS MAC Header identifies the SA and accompanying keying material that is used to encrypt the PDU. The CM MUST NOT encrypt MAC Management messages, except REG-REQ-MP messages, unless they are part of a fragment (see Section 6.5, Registration Request (REG-REQ-MP) MAC Management Messages).

6.2 CMTS Requirements

When communicating with a CM for which DOCSIS security is enabled, the CMTS MUST encrypt the PDU regions of variable-length PDU MAC frames and variable-length Isolation PDU MAC frames [DOCSIS MULPIv4.0].

A Baseline Privacy EH element in the DOCSIS MAC Header identifies the SA and accompanying keying material that is used to encrypt the PDU. The CMTS MUST NOT encrypt MAC Management messages (see Section 6.5, Registration Request (REG-REQ-MP) MAC Management Messages).

6.3 Variable-Length PDU MAC Frame Format

Figure 4 depicts the format of a DOCSIS variable-length PDU MAC frame or variable-length Isolation PDU MAC frame, with a Baseline Privacy EH element and encrypted PDU payload.

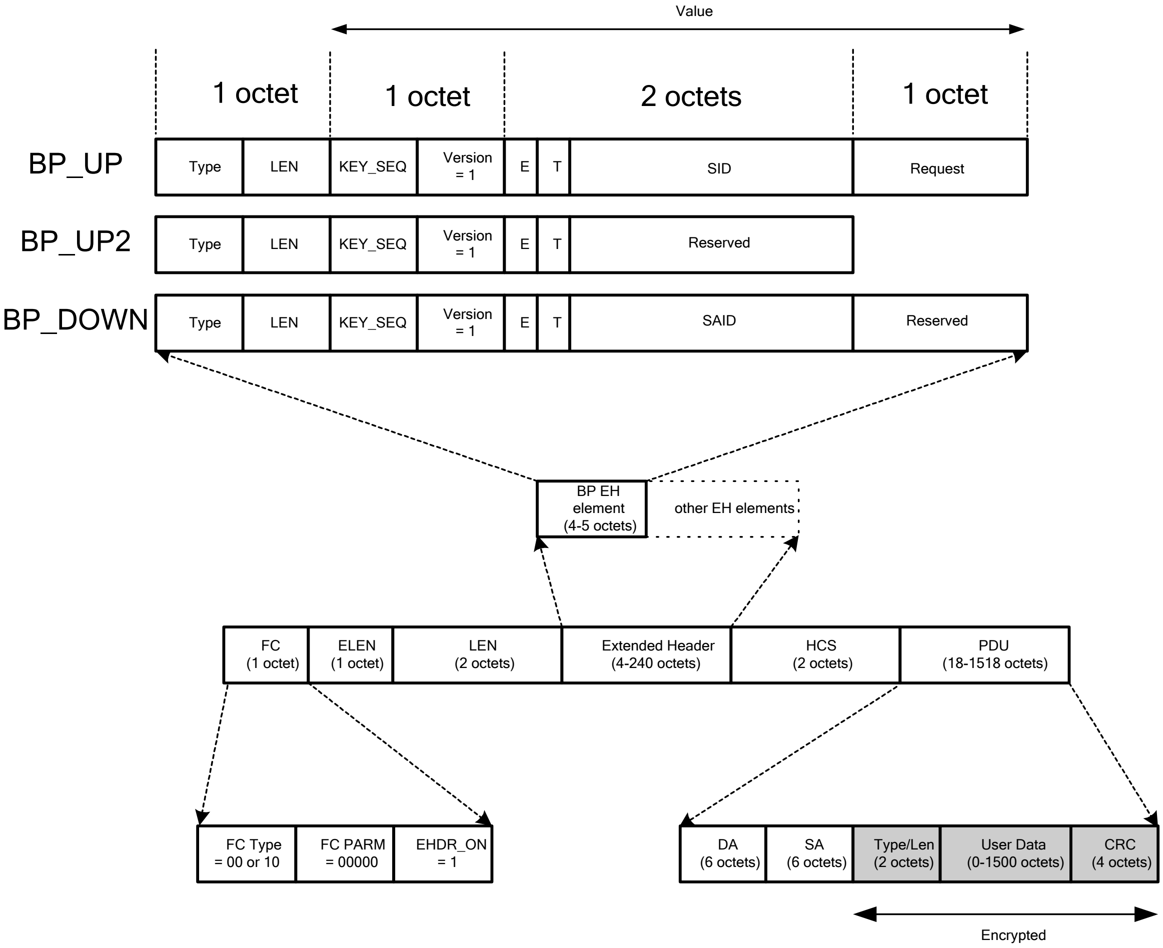

Figure 4 - Format of DOCSIS Variable-Length PDU with Baseline Privacy EH Element

The CMTS MUST NOT encrypt the first twelve (12) octets of the PDU containing the Ethernet/802.3 destination and source addresses (DA/SA). The CM MUST NOT encrypt the first twelve (12) octets of the PDU containing the Ethernet/802.3 destination and source addresses (DA/SA). The CMTS MUST encrypt the PDU's Ethernet/802.3 cyclic redundancy check (CRC). The CM MUST encrypt the PDU's Ethernet/802.3 CRC.

The CM MUST include a Baseline Privacy EH in all frames containing encrypted PDUs. The CM MUST make the Baseline Privacy EH element the first extended header that it sends in an upstream frame. Upstream frames sent by the CM MUST contain the Type value BP_UP or BP_UP2 in the Baseline Privacy EH.

The CMTS MUST include a Baseline Privacy EH in all frames containing encrypted PDUs. The Baseline Privacy EH element MUST be the first extended header sent by the CMTS in a downstream frame. Downstream frames sent by the CMTS MUST contain the Type value BP_DOWN [DOCSIS MULPIv4.0] in the Baseline Privacy EH. The high-order bits of a Baseline Privacy EH Value field contain a key sequence number, KEY_SEQ. The CMTS manages a key sequence number independently for each SAID and distributes this key sequence number along with the SAID's keying material to client CMs. For each SAID, the CMTS MUST increment KEY_SEQ by one each time that it generates new keying material for that SAID. The Baseline Privacy EH element includes this sequence number, along with the SAID, to identify the keying material that was used to encrypt the frame's PDU.

The four (4) bits following KEY_SEQ contain a protocol version number. The CMTS MUST set the protocol version number to the value one (1). The CM MUST set the protocol version number to the value one (1).

The next two octets contain two (2) bits of encryption status (ENABLE and TOGGLE) and the fourteen (14)-bit Reserved/SID/SAID (reserved for upstream frames with the BP_UP2 Extended MAC Header (EHDR), SID for upstream frames with the BP_UP EHDR, and SAID for downstream frames).

The ENABLE encryption status bit indicates whether encryption is enabled for that PDU. If the PDU is unencrypted, then the CMTS MUST set the ENABLE bit to zero (0). If the PDU is unencrypted, then the CM MUST set the ENABLE bit to zero (0). If the PDU is encrypted, then the CMTS MUST set the ENABLE bit to one (1). If the PDU is encrypted, the CM MUST set the ENABLE bit to one (1).

The CMTS MUST make the TOGGLE bit match the value of the Least Significant Bit (LSB) of KEY_SEQ. The CM MUST make the TOGGLE bit match the value of the Least Significant Bit (LSB) of KEY_SEQ.

6.3.1 Baseline Privacy Extended Header Formats

The DOCSIS MAC protocol [DOCSIS MULPIv4.0] defines a Request EH element that is used to piggyback a bandwidth request on an upstream data transmission. The last octet of the upstream Baseline Privacy EH (BP_UP) carries an optional piggybacked bandwidth allocation request.

In downstream packets, the last octet is reserved [DOCSIS MULPIv4.0] and the CMTS MUST set the value of the last octet in downstream packets to zero.

Table 3 - Summary of the Contents of Baseline Privacy EHs

Type	Length	Value
BP_UP See [DOCSIS MULPIv4.0]	4	KEY_SEQ (4 bits), Version (4 bits), SID (2 octets), Request [piggyback] (1 octet) [CM → CMTS] KEY_SEQ field (4 bits): Key sequence number Version field (4 bits) is defined as: 0x1 SID field is defined as: Bit[15]: ENABLE: 1⇒Encryption enabled; 0⇒Encryption Disabled Bit[14]: TOGGLE: 1⇒Odd Key; 0⇒Even Key Bit[13:0]: Service ID. Request field contains the number of minislots requested for upstream bandwidth.
BP_UP2 See [DOCSIS MULPIv4.0]	3	KEY_SEQ (4 bits), Version (4 bits), ETR (2 octets) [CM → CMTS] KEY_SEQ field (4 bits): Key sequence number Version field (4 bits) is defined as: 0x1 ETR field is defined as: Bit[15]: ENABLE: 1⇒Encryption enabled; 0⇒Encryption Disabled Bit[14]: TOGGLE: 1⇒Odd Key; 0⇒Even Key Bit[13:0]: Reserved (set to 0)
BP_DOWN See [DOCSIS MULPIv4.0]	4	KEY_SEQ (4 bits), Version (4 bits), SAID (2 octets), Reserved (1 octet) [CMTS → CM] KEY_SEQ field (4 bits): Key sequence number Version field (4 bits) is defined as: 0x1 SAID field is defined as: Bit[15]: ENABLE: 1⇒Encryption enabled; 0⇒Encryption Disabled Bit[14]: TOGGLE: 1⇒Odd Key; 0⇒Even Key Bit[13:0]: Security Association ID. Reserved field is set to 0.

6.4 Fragmentation MAC Frame Format

In order to support fragmentation of upstream DOCSIS MAC frames, the Baseline Privacy EH element may carry both encryption and fragmentation control fields [DOCSIS MULPIv4.0]. When functioning in this role, the upstream Baseline Privacy EH is extended by one octet, with the additional octet serving as a fragmentation control field. Figure 5 depicts the format of a DOCSIS Fragmentation MAC frame with an encrypted fragmentation payload.

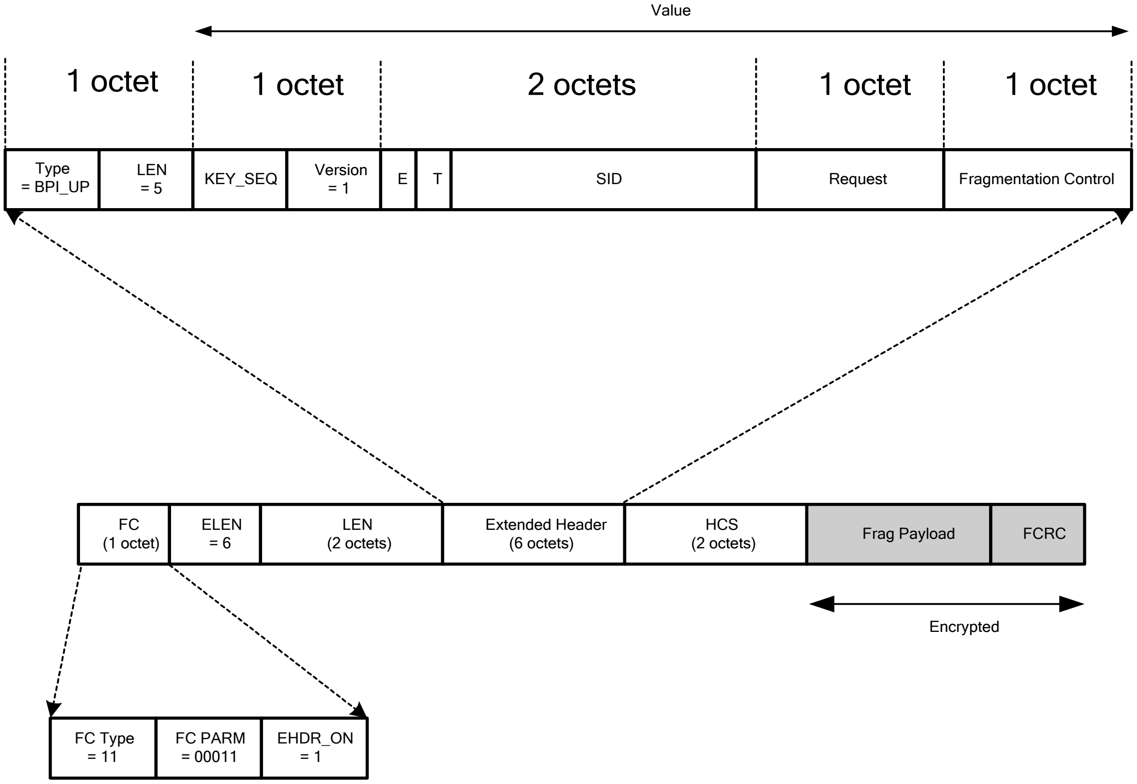

Figure 5 - Format of a DOCSIS MAC Fragmentation Frame with an Encrypted Payload

Frames with frame control (FC) Type set to 0b11 and FC PARM set to 0b00011 identify a DOCSIS MAC frame as a Fragmentation frame. The Fragmentation MAC header is followed by a Fragment Payload and a Fragment CRC. When encrypting a Fragmentation MAC frame, the CM MUST encrypt the Fragment Payload and the Fragment CRC (FCRC).

The LEN field of the Baseline Privacy EH element in Fragmentation MAC frames is five (5), rather than four (4), accounting for the additional one (1)-octet fragmentation control field. The definitions and requirements for the KEY_SEQ field, VERSION field, ENABLE and TOGGLE flags, and SID field are unchanged from those for an upstream PDU MAC frame.

Table 4 - Summary of the Contents of a DOCSIS Fragmentation MAC Frame's Baseline Privacy EH

Type	Length	Value
BP_UP See [DOCSIS MULPIv4.0]	5	KEY_SEQ (4 bits), Version (4 bits), SID (2 octets), Request [piggyback] (1 octet), Fragmentation Control (1 octet) [CM → CMTS] KEY_SEQ field (4 bits): Key sequence number Version field (4 bits) is defined as: 0x1 SID field is defined as: Bit[15]: ENABLE: 1⇒Encryption enabled; 0⇒Encryption Disabled Bit[14]: TOGGLE: 1⇒Odd Key; 0⇒Even Key Bit[13:0]: Service ID. Request field contains the number of minislots requested for upstream bandwidth. Fragmentation Control field contains fragmentation-specific control information; see [DOCSIS MULPIv4.0] for details.

The CM determines whether a packet will be fragmented based on its knowledge of the grant size (i.e., the number of minislots a CMTS grants to a CM in an Upstream Bandwidth Allocation MAP [DOCSIS MULPIv4.0]). If an encrypted packet is to be fragmented, the CM MUST perform encryption on a fragment-by-fragment basis, not over the PDU as a whole; each fragment will, therefore, have its own fragmentation header and be encrypted separately.

6.5 Registration Request (REG-REQ-MP) MAC Management Messages

When EAE is enabled, the CM MUST encrypt REG-REQ-MP MAC management messages (see Section 9.5, Securing REG-REQ-MP Messages). Figure 6 depicts the format of a DOCSIS MAC management message frame with a Baseline Privacy EH element and encrypted payload.

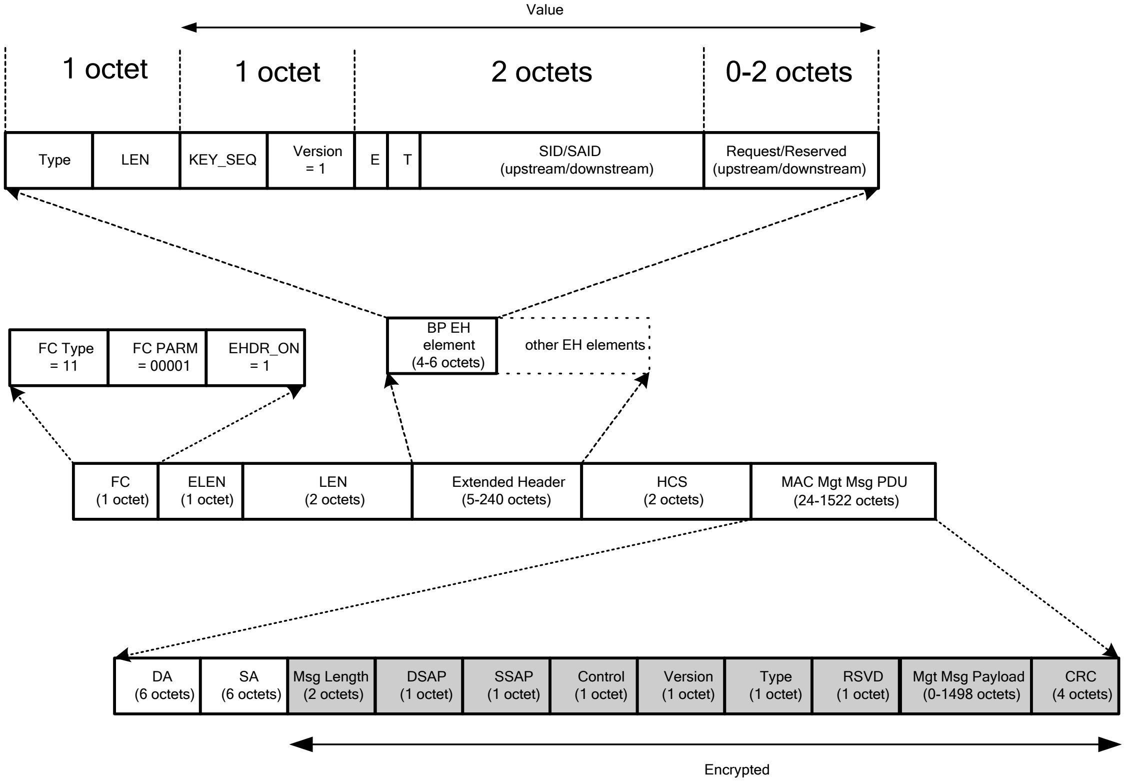

Figure 6 - Format of a DOCSIS MAC Management Message Frame with Encrypted Payload

The CM MUST encrypt REG-REQ-MP MAC management message frames according to the following rules:

- The first twelve (12) octets of the PDU, containing the Ethernet/802.3 destination and source addresses (DA/SA) MUST NOT be encrypted;
- The CM MUST encrypt the PDU's Ethernet/802.3 CRC;
- A Baseline Privacy EH MUST be included in the frame;
- The Baseline Privacy EH element MUST be the first extended header in the upstream frame;
- The upstream frames MUST contain the Type value BP_UP in the Baseline Privacy EH;
- The four (4) bits following KEY_SEQ contain a protocol version number; the protocol version number MUST have the value one (1);
- The next two octets contain two (2) bits of encryption status (ENABLE and TOGGLE) and the fourteen (14) bit SID; the ENABLE bit MUST have the value one (1);
- The TOGGLE bit MUST match the value of the Least Significant Bit (LSB) of KEY_SEQ.

Table 5 - Summary of the Contents of DOCSIS MAC Management Message Baseline Privacy EHs

Type	Length	Value
BP_UP See [DOCSIS MULPIv4.0]	4	KEY_SEQ (4 bits), Version (4 bits), SID (2 octets), Request [piggyback] (1 octet) [CM → CMTS] KEY_SEQ field (4 bits): Key sequence number Version field (4 bits) is defined as: 0x1 SID field is defined as: Bit[15]: ENABLE: 1⇒Encryption enabled; 0⇒Encryption Disabled Bit[14]: TOGGLE: 1⇒Odd Key; 0⇒Even Key Bit[13:0]: Service ID. Request field contains the number of minislots requested for upstream bandwidth.
BP_UP2 See [DOCSIS MULPIv4.0]	3	KEY_SEQ (4 bits), Version (4 bits), SID (2 octets) [CM → CMTS] KEY_SEQ field (4 bits): Key sequence number Version field (4 bits) is defined as: 0x1 SID field is defined as: Bit[15]: ENABLE: 1⇒Encryption enabled; 0⇒Encryption Disabled Bit[14]: TOGGLE: 1⇒Odd Key; 0⇒Even Key Bit[13:0]: Service ID.
BP_UP2 See [DOCSIS MULPIv4.0]	5	KEY_SEQ (4 bits), Version (4 bits), SID (2 octets), Request [piggyback] (2 octets) [CM → CMTS] KEY_SEQ field (4 bits): Key sequence number Version field (4 bits) is defined as: 0x1 SID field is defined as: Bit[15]: ENABLE: 1⇒Encryption enabled; 0⇒Encryption Disabled Bit[14]: TOGGLE: 1⇒Odd Key; 0⇒Even Key Bit[13:0]: Service ID. Request field contains the number of octets requested for upstream bandwidth.

6.6 Use of the Baseline Privacy Extended Header in the MAC Header

If encryption is not enabled on a particular downstream traffic flow (e.g., a CM's unicast traffic or a particular IP multicast group), then the CMTS SHOULD NOT place a Baseline Privacy EH element in the frame.

If encryption is not enabled for a CM's unicast traffic, then the CM MUST include the Baseline Privacy EH element with the ENABLE bit set to 0 in fragmented upstream frames.

If a CM sends a MAC frame consisting of only a MAC header and, optionally, an EHDR, then the CM MUST disable encryption on that frame. If a Baseline Privacy EHDR is present on such a frame, then the CM MUST set the ENABLE bit to zero (0).

If a CMTS sends a MAC frame consisting of only a MAC header and, optionally, an EHDR, then the CMTS MUST disable encryption on that frame. If a Baseline Privacy EHDR is present on such a frame, then the CMTS MUST set the ENABLE bit to zero (0).

7 BASELINE PRIVACY KEY MANAGEMENT (BPKM) PROTOCOL

7.1 State Models

7.1.1 Introduction

The BPKM protocol is controlled by two separate but interdependent state machines, described in this section: an Authorization state machine and a TEK state machine. The state machines are presented here for explanatory purposes only and are not to be construed as constraining an actual implementation. However, the external behavior of CMTS implementations MUST be identical to the state machines described in this section. The external behavior of CM implementations MUST be identical to the state machines described in this section.

CM authorization, controlled by the Authorization state machine, uses the following process:

- the CMTS authenticates a client CM's identity by validating the signature on the Authorization Request message (BPI+ V2 only) and by verifying the certificate chain of the device certificate used to sign the message with;
- the CMTS provides the authenticated CM with ephemeral Key Exchange parameters (BPI+ V2) or with an Authorization Key (BPI+ V1) in the Authorization Reply ("Auth Reply") message, from which a KEK and the message authentication keys are derived;
- the CMTS provides the authenticated CM with the identities (i.e., the SAIDs) and properties of Primary and Static SAs for which the CM is authorized to obtain keying information; and
- the CM authenticates the received parameters by verifying the message signature and the certificate chain from the serving CMTS and sends an Auth Status Info message to the CMTS that contains the result code of the CMTS authentication performed by the CM (BPI+ V2 only).

The KEK is a two-key, 3DES encryption key that the CMTS uses to encrypt TEKs that it sends to the CM. TEKs are used to encrypt and decrypt user data traffic and REG-REQ-MP MAC management messages. The CM and CMTS use message authentication keys to authenticate, via a keyed message digest, the key requests and the responses that they exchange.

After achieving initial authorization, a CM periodically seeks reauthorization with the CMTS; reauthorization is managed by the CM's Authorization state machine. A CM maintains its authorization status with the CMTS in order to be able to refresh TEKs. TEK state machines manage the refreshing of Traffic Encryption Keys.

7.1.1.1 Authorization State Machine Overview

A CM begins the authorization process by sending an Authentication Information message to its CMTS. The Authentication Information message contains the device's issuing CA Certificate. The Authentication Information message is usually informative (i.e., the CMTS may ignore it under some circumstances); however, it does provide a mechanism for a CM to communicate intermediate CA certificates to the CMTS (see Section 7.1.6.2.5 for more details).

The CM sends an Authorization Request message to the CMTS. This is a request for an Authorization Key and for the SAIDs that identify any Static SAs in which the CM is authorized to participate. The CM MUST include the following common attributes in the Authorization Request message for both BPI+ V1 and BPI+ V2:

- A description of the security and cryptographic algorithms supported by the CM (see Section 7.2.2.19, Security-Capabilities). A CM's cryptographic capabilities are presented to the CMTS as a list of cryptographic suite identifiers, each indicating a particular pairing of packet data encryption and packet data authentication algorithms, supported by the CM.
- The Initialization SAID (see Section 7.2.2.12, SAID). The CM MUST use a value of zero for the Initialization SAID. The CMTS MUST interpret an SAID value of zero as an Initialization SAID.
- Authorization Request messages additionally include several BPI+ version-specific fields.

When BPI+ V1 is negotiated (see Section 7.1.3, BPI+ Versioning), the CM MUST additionally include the following fields:

- the CM's identification information (see Section 7.2.2.5, CM-Identification),
- a CM Device Certificate (see Section 7.2.2.18, CM-Certificate) binding the CM's public key to its other identifying information. This means that, when using BPI+ V1, the CM public key appears twice in the message, once as a standalone entity and once as part of the certificate.

When BPI+ V2 is negotiated (see Section 7.1.3, BPI+ Versioning), the CM MUST additionally include the following fields:

- the BPI+ version (2), (see Section 7.2.2.22, BPI-Version)
- the CM's serial number, (see Section 7.2.2.1, Serial-Number)
- the CM's manufacturer's ID, (see Section 7.2.2.2, Manufacturer-ID)
- a NONCE value (see Section 7.2.2.33, CM-Nonce)

  The NONCE value is a randomly generated value that is provided by the CM to make sure that replies from the CMTS are part of the same authorization exchange (and not replayed from a previous exchange).
- a key exchange share (see Section 7.2.2.35, Key-Exchange-Share)
- a message signature attribute (see Section 7.2.2.34, Message-Signature)

  In BPI+ V2, Authorization Request messages are authenticated by using a detached CMS signature that is calculated over all the attributes present in the message in the order they appear in the message definition (excluding the signature attribute itself).

The CM MUST include its own ephemeral Elliptic Curve Diffie-Hellman parameters (ECDHE) to enable mutual key derivation and Perfect Forward Security.

The CM enables authentication of the Authorization Request message by generating a digital signature with its own Device Certificate as described in 7.1.1.2.3.

Upon receiving an Authorization Request message, the CMTS MUST

- check the BPI+ version number,
- verify the `Message-Signature` attribute (BPI+ Version 2 only),
- validate the CM Device Certificate and its Certificate Chain,
- check that the CMTS and the CM share at least one cryptographic suite, and
- assign the CM's Primary SAID.

In addition, the CMTS MUST also verify that the MAC-address attribute of the Auth Request (for BPI+ V1 only), the MAC address in the CM's Device certificate (BPI+ V2), and the source MAC address (BPI+ V1 and V2) of the Authorization Request message all match.

In BPI+ V2, Authorization Request messages are authenticated using a detached CMS signature that contains the CM device certificate.

When using BPI+ V2, the CMTS MUST verify the authenticity of the Authorization Request message before accepting the request as described in Section 7.1.1.2.4, Validation of CM Messages Authentication.

The CMTS SHOULD, by means that are outside the scope of this specification, determine whether the CM is authorized to receive service and, if so, whether it is entitled to access services that are served by any Static SAs.

If these conditions are met, the CMTS MUST then either create (BPI+ Version 1) or derive (BPI+ Version 2) the keying material for communicating with the CM.

When using BPI+ V1, the CMTS MUST generate an Authorization Key for the CM and encrypt it with the CM's public key.

When using BPI+ V2, the CMTS MUST both generate its own key exchange parameters and derive the Authorization key by combining such parameters with the public ones received from the CM.

The CM MUST use its own parameters combined with the CMTS's public ones to derive the same Authorization Key.

In BPI+ V2, the CMTS MUST enable authentication of the reply message, i.e., the Authorization Reply or the Authorization Reject message, by generating the message signature as described in Section 7.1.1.2.1, CMTS Message Authentication.

If the CMTS is able to authenticate the Authorization Request message from the CM but does not authorize the CM for operation, the CMTS MUST send an Authorization Reject message to the CM containing the following attributes:

- an Error Code, indicating the reason the CMTS is rejecting the CM, (see Section 7.2.2.15, Error-Code),
- a Display String, which can include additional human-readable details about the rejection reason.

When BPI+ V2 is negotiated and not authorized, the CMTS MUST additionally include these attributes in the Authorization Reject message (see Section 7.2.1.3, Authorization Reject (Auth Reject)):

- the BPI+ version (see Section 7.2.2.22, BPI-Version),
- the NONCE value provided by the CM in the Authorization Request (see Section 7.2.2.33, CM-Nonce),
- an optional list of OCSP responses for the CMTS certificate chain (see Section 7.2.2.37, OCSP-Responses), and
- a message signature together with the corresponding signing CMTS Device and Issuer certificates (see Section 7.2.2.34, Message-Signature).

The CMTS includes the OCSP-Responses Attribute (see Section 7.2.2.37, OCSP-Responses) to convey the revocation status for the CMTS certificate and its Issuing CA.

If the CMTS supports Static SAs, then the CMTS MUST include the identities of all Static SAs associated with the CM, in addition to the Primary SA, in the CMS signature.

The CMTS MUST NOT identify any Dynamic SAs (see Section 7.1.2, Encrypted Multicast) in the Authorization Reject.

If the CMTS authorizes the CM for operation, the CMTS MUST send an Authorization Reply message to the CM containing the following common attributes (BPI+ V1 and BPI+ V2):

- a four (4) bit key sequence number (see Section 7.2.2.10 Key-Sequence-Number),
- a key lifetime (see Section 7.2.2.9 Key-Lifetime), and
- the identities (i.e., the SAIDs) and properties of the Primary SA and zero or more Static SAs for which the CM is authorized to obtain keying information (see Section 7.2.2.23 SA-Descriptor)

If BPI+ V1 is negotiated, the Authorization Reply message, in addition to the common attributes, additionally contains:

- The Authorization Key encrypted with the CM's public key (see Section 7.2.2.7, Auth-Key).

If BPI+ V2 is negotiated, the Authorization Reply message additionally includes:

- the BPI+ version (2), (see Section 7.2.2.22, BPI-Version),
- an optional specification of the BPI+ version restrictions (see Section 7.2.2.36, Allowed-BPI-Versions),
- the NONCE value provided by the CM in the Authorization Request, (see Section 7.2.2.33, CM-Nonce),
- an optional specification of the CMTS designation (see Section 7.2.2.38, CMTS-Designation),
- an optional list of OCSP responses for the CMTS certificate chain (see Section 7.2.2.37, OCSP-Responses),
- the CMTS' key exchange public parameters (see Section 7.2.2.35, Key-Exchange-Share), and
- a message signature together with the corresponding signing CMTS Device and Issuer certificates (see Section 7.2.2.34, Message-Signature).

If the CMTS supports Static SAs, then the CMTS MUST include in the Authorization Reply message the identities of all Static SAs associated with the CM, in addition to the Primary SA.

The CMTS MUST NOT identify any Dynamic SAs (see Section 7.1.2, Encrypted Multicast) in the Authorization Reply message.

NOTE WELL: BPI+ V2 introduces a new method for generating the Authorization Key. Specifically, ephemeral key exchange parameters are used in the authorization message exchange to enable the CMTS and the CM to derive the Authorization Key instead of directly encrypting it for the CM via the `rsaEncryption` field. The new mechanism enables Perfect Forward Secrecy (PFS) and Algorithm Agility.

To generate the signature, the CMTS follows exactly the same procedures used to generate for the Authorization Reject message signature as described in Section 7.1.1.2.1.

The CMTS includes the `OCSP-Responses` Attribute (see Section 7.2.2.37) to convey the revocation status for the CMTS certificate and its Issuing CA.

The CMTS MAY include the `Allowed-BPI-Versions` (see Section 7.2.2.36, Allowed-BPI-Versions) and CMTS-Designation (see Section 7.2.2.38, CMTS-Designation) Attributes to specify connection parameters (i.e., the allowed BPI+ Versions and CMTS' certificate filtering) that the CM is required to validate honor during future authorization exchange(s).

Upon receiving an Authorization Reply, the CM MUST verify the `Message-Signature,` validate the signer's certificate chain, and check the CMTS-Designations values (see Section 7.1.1.2.2, Validation of CMTS Messages Authentication).

The CM MUST discard any Authorization Reply messages that fail signature verification.

If the CM successfully validates all the elements of the Authorization Reply message, the CM MUST store the `Allowed-BPI-Versions` and `CMTS-Designation` Attributes from the Authorization Reply message in the Persistent Security Attributes, if/when present. In BPI+ V2 only, the CM sends the result code of the CMTS authentication back to the CMTS in an Authorization Status Info message, which enables the CMTS to detect any authentication issues that might have occurred on the CM.

> NOTE WELL: When enabling EAE, the Authorization Status Info message provides the means for the CM to return any authentication issue that might happen, thus enabling the possibility to early detect authentication and authorization issues.

The CM MUST start a separate TEK state machine for each of the SAIDs identified in the Authorization Reply message (see Section 7.2.1.2, Authorization Reply (Auth Reply)). Dynamic SAs are not included in the Authorization Reply and, therefore, are not shown in Figure 7.

If the CM processes a BPI+ V2 Authorization Reply message, the CM MUST issue an Authorization Status Info message (see Section 7.2.1.13, Authentication Status Info (Auth Status Info)) which includes the following fields attributes:

- the BPI+ version (see Section 7.2.2.22, BPI-Version),
- a NONCE value (see Section 7.2.2.33, CM-Nonce) which is a randomly generated value that is provided by the CM to make sure that replies from the CMTS are part of the same authorization exchange and not replayed from a previous exchange,
- a status code indicating the CM's authorization status (see Section 7.2.2.39, CM-Status-Code),
- an optional display string providing a human-readable authorization status (see Section 7.2.2.6, Display-String),
- an optional numerical count of the number of detected authentication issues (see Section 7.2.2.40, Detected-Errors), and
- a message signature attribute (see Section 7.2.2.34, Message-Signature).

The CM MUST indicate either authentication success or the primary reason for authentication failure using the `CM-Status-Code` Attribute (see Section 7.2.2.39, CM-Status-Code).

The CM SHOULD additionally include the Display-String attribute (see Section 7.2.2.6, Display-String) with additional human-readable details.

The CM SHOULD detect as many authentication issues as possible and provide a count of any additional issues in the `Detected-Errors` Attribute (see Section 7.2.2.40, Detected-Errors).

The CM MUST calculate the Message-Signature (see Section 7.2.2.34, Message-Signature) over the BPKM message header (starting with the BPKM Code field) as described in Section 7.1.1.2.3, CM Message Authentication.

Upon receiving an Authorization Status Info message, the CMTS MUST first validate the `Message-Signature` as described in Section 7.1.1.2.4, Validation of CM Messages Authentication.

If the Authorization Status Info message is successfully validated, the CMTS can either report the CMs authorization status to the operator to indicate successful authentication by the CM or to diagnose authentication issues if/when they occur.

7.1.1.2 Authorization Messages Authentication Overview

7.1.1.2.1 CMTS Message Authentication

When message authentication is needed, e.g., in either the Authorization Reply message or the Authorization Reject messages, the CMTS MUST generate a `Message-Signature` attribute and include it in the authenticated message itself.

The CMTS MUST calculate the Message-Signature Attribute (see Section 7.2.2.34, Message-Signature) over the BPKM message header starting with the BPKM Code field and all of the attributes in the message, other than the message signature itself, in the same order as they appear in the message and place the message signature at the end of the attribute list.

The CMTS MUST use the private key associated with its device certificate to calculate the signature.

The CMTS MUST include its own certificate in the certificates field of the CMS SignedData structure.

The CMTS MUST include its own device issuing CA certificate in the certificates field of the CMS SignedData structure.

The CMTS MUST include the OCSP responses in the `OCSP-Responses` Attribute of the message to provide the revocation status of its own certificate and certificate chain (see Section 7.2.2.37, OCSP-Responses).

The CMTS MAY omit the OCSP responses for certificates in its chain if and only if the URL of the OCSP responder is not present in the certificate's Authority Info Access extension of the certificate.

7.1.1.2.2 Validation of CMTS Messages Authentication

Upon receiving an authenticated message from the CMTS (e.g., an Authorization Reply or Authorization Reject), the CM verifies the `Message-Signature` first.

Specifically, the CM MUST verify the signature over the BPKM message header starting with the BPKM Code field and all the attributes in the message, other than the message signature itself, in the same order as they appear in the message.

After verifying the message signature, the CM MUST then validate the CMTS certificate chain included in the message signature and reject messages if the certificate chain is not trusted, i.e., no certificates in the chain are considered as trusted on the CM.

The CM MUST check for the presence of the `id-cl-pki-ext-eku-CMTS` OID in the Extended Key Usage (EKU) in the CMTS certificate (see Appendix III.1.1).

If either the EKU extension is not present in the CMTS certificate or the `id-cl-pki-ext-eku-CMTS` OID is not present among the EKU values, the CM MUST reject the message as not trusted.

The CM MUST discard any Authorization Reply messages that fail signature verification.

If the OCSP-Responses Attribute is present, the CM MUST validate the revocation status of the CMTS Certificate and its chain by utilizing the OCSP-Responses Attribute and rejecting the message if any of the certificate present in the chain has been revoked.

If the URL of the authoritative OCSP responder is not present in the CMTS certificate or in the CMTS CA certificate, the CM MAY omit checking the OCSP responses for that certificate.

The CM MUST fail the validation of the CMTS certificate if any of the OCSP-Responses for the CMTS certificate chain cannot be successfully authenticated or correctly processed.

The CM MUST validate the CMTS certificate (and/or CMTS' Issuing CA certificate) meets the criteria of the CMTS-Designation when specific configurations are present in the Persistent Security Attribute (see Section 12.2, Persistent Security Attribute Values).

The CM MUST discard any signed messages that fail these validation steps.

7.1.1.2.3 CM Message Authentication

When message authentication is needed (e.g., in Authorization Request or Authorization Status Info messages), the CM MUST generate a Message-Signature attribute and include it in the authenticated message itself.

The CM MUST calculate the Message-Signature (see Section 7.2.2.34, Message-Signature) over the BPKM message header starting with the BPKM Code field and all of the attributes in the message, other than the message signature itself, in the same order as they appear in the message and place the signature at the end of the attribute list.

The CM MUST use the private key associated with its device certificate to calculate the signature.

The CM MUST add its own certificate to the certificates field of the CMS SignedData structure.

The CM MAY add its own issuer CA's certificate to the certificates field of the CMS SignedData structure.

7.1.1.2.4 Validation of CM Messages Authentication

Upon receiving an authenticated message (e.g., an Authorization Request), the CMTS verifies the Message-Signature first.

The CMTS MUST verify the Message-Signature over the BPKM message header starting with the BPKM Code field and all the attributes in the message, other than the message signature itself, in the same order as they appear in the message.

The CMTS MUST check for the presence of the id-cl-pki-ext-eku-CM OID in the Extended Key Usage (EKU) in the CM certificate (see Appendix III.1.1).

If either the EKU extension is not present in the CM certificate or the id-cl-pki-ext-eku-CM OID is not present among the EKU values, the CMTS MUST reject the message as not trusted.

In case the URL of the authoritative OCSP responder is available in the CM Device Certificate and/or in the CA Certificate or via configuration options, the CMTS MUST query the specific OCSP responder and check the revocation status of the CM Device Certificate and its certification chain by following the procedures identified in Section 13.3, Certificate Processing and Management in the CMTS.

The CMTS MUST discard any signed messages that fail these validation steps.

7.1.1.3 TEK State Machine Overview

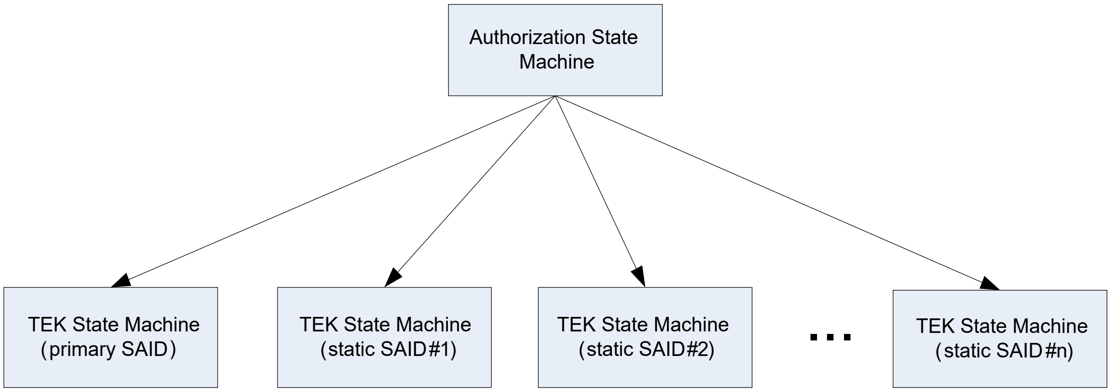
Authorization State Machine
TEK State Machine (primary SAID)
TEK State Machine (static SAID#1)
TEK State Machine (static SAID#2)
...
TEK State Machine (static SAID#n)

Figure 7 - Relationship among Authorization and TEK State Machines

Each TEK state machine is responsible for managing the keying material associated with its respective SAID. TEK state machines send Key Request messages to the CMTS, requesting initial and subsequent keying material for their respective SAIDs. A Key Request contains

- identifying information unique to the CM. When BPI+ V1 is used, the identifying information includes the manufacturer ID, serial number, MAC address, and the RSA Public Key. When BPI+ V2 is used, the identifying information includes the CM’s MAC address;
- the SAID whose keying material is being requested; and
- an HMAC keyed message digest, authenticating the Key Request.

The CMTS checks the HMAC digest of the Key Request message (see Sections 11 and 12). If the CMTS verifies the HMAC, then the CMTS MUST respond with a Key Reply message containing the CMTS's active keying material for the specific SAID. This keying material includes

- the 3DES-encrypted TEK,
- a CBC initialization vector,
- the key sequence number,
- the key's remaining lifetime, and
- an HMAC keyed message digest.

The TEK in the Key Reply is 3DES (EDE mode) encrypted, using a two-key, 3DES KEK derived from the Authorization Key (see Section 11.2).

At all times, the CMTS maintains two active sets (called "generations") of keying material for each SAID. A CMTS includes in its Key Replies both of an SAID's currently valid generations of keying material.

In addition to the TEK and CBC initialization vector, the Key Reply contains the remaining lifetime of each of the two sets of keying material. The CM uses these remaining lifetimes to estimate when to schedule Key Requests, such that the CM requests and receives new keying material before the CMTS invalidates the keying material currently held by the CM.

The operation of the TEK state machine's Key Request scheduling algorithm, combined with the CMTS's process for updating and using an SAID's keying material, ensures that the CM will be able at all times to exchange encrypted traffic with the CMTS.

Before the current Authorization Key expires, the CM obtains a new Authorization Key by issuing an Authorization Request to the CMTS. This "reauthorization" is identical to authorization, with the exception that the CM does not send an Authentication Information message. The specification of the Authorization state machine defines when Authentication Information messages are sent (see Section 7.1.6).

To avoid service interruptions during reauthorization, successive generations of the CM's Authorization Keys have overlapping lifetimes.

The CM MUST be able to support at least two simultaneously active Authorization Keys.

The CMTS MUST be able to support at least two simultaneously active Authorization Keys for each registered CM.

The operation of the Authorization state machine's Authorization Request scheduling algorithm, combined with the CMTS's process for updating and using a CM's Authorization Keys, ensures that CMs will be able to refresh TEK keying information without interruption over the course of the CM's reauthorization periods.

A TEK state machine remains active as long as

- the CM has a valid Authorization Key and
- the CMTS continues to provide fresh keying material during TEK re-key cycles.

The parent Authorization state machine stops all its child TEK state machines (see Figure 7), when the CM receives an Authorization Reject during a reauthorization cycle. Individual TEK state machines can be started or stopped during a reauthorization cycle if a CM's Static SAID authorizations change between successive reauthorizations.

Communication between Authorization and TEK state machines occurs through the passing of events (directly or indirectly) between the two state machines. The Authorization state machine generates events ({Stop}, {Authorized}, {Authorization Pending}, and {Authorization Complete}) that are sent directly to all its child TEK state machines. TEK state machines, however, cannot directly send events to their parent Authorization state machine.

A TEK state machine affects its parent Authorization state machine indirectly through the messaging a CMTS sends in response to a CM's requests: a CMTS may respond to a TEK machine's Key Requests with a failure response (i.e., an Authorization Invalid message) that will be handled by the Authorization state machine. In other words, the TEK state machine might transmit a Key Request to the CMTS, which may respond with an Authentication Invalid message. This message is handled not by the TEK state machine responsible for transmitting the Key Request but by its parent Authorization state machine.

7.1.2 Encrypted Multicast

The message exchange between the CMTS and the CM for the signaling and initialization of multicast traffic encryption is dependent on the type of multicast session, on the capabilities of the modem, and on the multicast forwarding mode selected by the CMTS. The CMTS selects a multicast forwarding mode within the multicast forwarding capabilities reported by the CM.

Multicast sessions can be established dynamically when a Multicast Client sends a join request message (IGMP for IPv4 and MLD for IPv6). Such multicast sessions are called Dynamically joined Multicast Sessions. The cable operator can also configure the cable modem to join multicast sessions during registration. Such multicast sessions are called Static Multicast Sessions [DOCSIS MULPIv4.0].

A CM indicates support for Downstream Service Identifier (DSID) Multicast Forwarding in the Registration Request message with a Multicast DSID Forwarding (MDF) capability encoding. A CM that reports in this encoding the value of either 1 (GMAC-Explicit) or 2 (GMAC-Promiscuous) is said to support MDF. A CM that omits this encoding or that reports in it a value of 0 (No Support for Multicast DSID Forwarding) is said not to support MDF. The CMTS enables MDF at a CM that supports MDF by setting the MDF capability encoding to the value 1 or 2 in the Registration Response it sends to the modem; such a CM is said to be an MDF-enabled CM. The CMTS may disable MDF at a CM that supports MDF by setting the MDF capability encoding to the value 0 in the Registration Response it sends to the modem [DOCSIS MULPIv4.0]; such a CM is said to be an MDF-disabled CM.

SAs are used to support encrypted multicast sessions. SA descriptors containing the SAID, the SA type, and cryptographic suite for encrypted multicast sessions are included in Auth Reply, REG-RSP(-MP), and DBC-REQ MAC messages and are used by the CM to create or delete the corresponding TEK state machines (see [DOCSIS MULPIv4.0]). The CMTS typically communicates in REG-RSP Dynamic SAs associated with static multicast sessions. The CMTS typically communicates in DBC-REQ Dynamic SAs associated with multicast sessions that are explicitly joined via multicast management protocols such as IGMP/MLD. Note that a REG-RSP may both enable DSID Multicast Forwarding and include added SAs.

7.1.2.1 Signaling of Dynamic and Static Multicast Session SAs when MDF is Disabled

If a CM does not support Multicast DSID Forwarding, then the CMTS MUST NOT signal Dynamic SAs to the CM in a REG-RSP or DBC-REQ message. For CMs that do not support MDF, the CMTS MUST signal SAs using the DOCSIS 1.1/2.0 Dynamic Security Association mechanism described in Annex C, DOCSIS 1.1/2.0 Dynamic Security Associations (Normative).

If the CMTS disables MDF for a CM that supports MDF, then the CMTS MUST NOT signal SAs used for encrypted multicast sessions using DBC-REQ to this CM.

The CMTS may signal SAs for other purposes to MDF-disabled CMs using DBC-REQ or REG-RSP(-MP).

The CM MUST accept SA Descriptor Encodings in REG-RSP(-MP), even if this message disables MDF.

If the CMTS disables MDF on a CM that supports MDF, then the CMTS MUST signal SAs for encrypting IP multicast traffic to this CM using the DOCSIS 1.1/2.0 Dynamic Security Association mechanism described in Annex C, DOCSIS 1.1/2.0 Dynamic Security Associations (Normative).

Note that a CMTS may communicate Static SAs to an MDF-disabled CM in the BPI+ Auth Reply. Such SAs may be used for encrypting static multicast sessions. The CM accepts Static SAs in BPI+ Auth Reply messages, even when it operates in MDF-disabled mode.

7.1.2.2 Signaling of Dynamic and Static Multicast Session SAs when MDF is Enabled

When the CMTS enables Multicast DSID Forwarding for a CM in the REG-RSP(-MP) [DOCSIS MULPIv4.0], the CM MUST NOT transmit Dynamic SA MAP Requests to the CMTS. The CMTS MUST respond with an SA Map Reject message containing error code 7 if it receives an SA Map Request from an MDF-enabled CM.

A CMTS MAY signal in a DBC-REQ the deletion of Dynamic SAs known to the CM. When the MDF-enabled CM receives a DBC-REQ that deletes a Dynamic SA, the CM MUST terminate the corresponding TEK state machine prior to sending the DBC-RSP and remove the Dynamic SA's keying material from the CM's key table.

The CM MUST discontinue decryption on an SA deleted through a DBC-REQ message. A CM MUST indicate an error response to an attempt to delete an unknown SA.

For encrypting multicast sessions (static or dynamic) forwarded through the MDF-enabled CM, the CMTS MUST use SAs ONLY of type "Dynamic".

The CMTS MUST NOT signal SAs for multicast sessions in BPI Auth Reply messages to an MDF-enabled CM.

The CMTS is allowed to signal SAs in BPI Auth Reply messages to an MDF-enabled CM for purposes other than multicast encryption. The CM accepts Static SAs in BPI+ Auth Reply messages, even when it operates in MDF-enabled mode.

7.1.2.2.1 Requirements Specific to the Signaling of Dynamic SAs for Dynamic Multicast Sessions

SA descriptors for encrypted multicast sessions joined dynamically are communicated to the MDF-enabled CM in a DBC-REQ message.

If a dynamic multicast session is encrypted, then the CMTS MUST communicate in a DBC-REQ message the session SA Descriptor to an MDF-enabled CM.

The CMTS MUST set the SA Type in the DBC-REQ message to "Dynamic" for a dynamic multicast session.

A CM for which the CMTS has enabled DSID Multicast Forwarding MUST accept in a DBC-REQ one or more Security Association Encodings that add a new SA of type "Dynamic".

The CMTS MUST NOT send a Dynamic SA in DBC-REQ messages if BPI+ is disabled for a CM.

When an authorized CM receives a DBC-REQ that contains a Dynamic SA, the CM MUST start a TEK state machine for the Dynamic SA prior to sending a DBC-RSP message.

The CMTS MUST NOT send a DBC-REQ with a Dynamic SA to a CM that is not in the "Authorized" state.

If an unauthorized CM receives a DBC-REQ message that adds a Dynamic SA, then the CM MUST reject the DBC-REQ message.

The CMTS is allowed to send a DBC-REQ with an SA that employs a cryptographic suite unsupported by the CM.

If an authorized CM receives a DBC-REQ message that adds a Dynamic SA having an unsupported cryptographic suite, then the CM MUST reject the DBC-REQ message.

If a CM receives a DBC-REQ message adding a Dynamic SA, in which the TEK state machine for that Dynamic SA is already active, then the CM MUST reject the DBC-REQ message.

7.1.2.2.2 Requirements Specific to the Signaling of Dynamic SAs for Static Multicast Sessions

SA descriptors for encrypted multicast sessions joined statically are communicated to the MDF-enabled CM in the REG-RSP(-MP) message.

If the static multicast session is encrypted, then the CMTS MUST communicate in REG-RSP(-MP) the session SA Descriptor to an MDF-enabled CM.

The CMTS MUST set the SA Type in the REG-RSP(-MP) message to "Dynamic" for a static multicast session.

A CM for which the CMTS has enabled DSID Multicast Forwarding MUST accept in a REG-RSP(-MP) one or more Security Association Encodings that add a new SA of type "Dynamic".

The CMTS MUST NOT send a Dynamic SA in REG-RSP(-MP) if BPI+ is disabled for a CM.

When an authorized CM receives a REG-RSP(-MP) that adds a Dynamic SA, the CM MUST start a TEK state machine for that Dynamic SA.

When an unauthorized CM receives a REG-RSP(-MP) that adds a Dynamic SA, the CM MUST wait until it reaches its Authorized state before starting a TEK state machine for that Dynamic SA.

The CMTS is allowed to send a REG-RSP(-MP) with an SA that employs a cryptographic suite unsupported by the CM.

If the CM receives a REG-RSP(-MP) message that adds a Dynamic SA having an unsupported cryptographic suite, then the CM MUST reject the REG-RSP(-MP) message.

7.1.3 BPI+ Versioning

A CM MUST be able to operate with a CMTS operating with BPI+ V1 and/or BPI+ V2, subject to the configuration of the Persistent Security Attributes (see Section 12.2, Persistent Security Attribute Values). The CM MUST attempt to determine the BPI versions and their configurations (i.e., features) that are enabled on the CMTS by inspecting the MAC Domain Descriptor (MDD) message TLV type 23 which is carried on all primary-capable channels of a downstream service group. See [DOCSIS MULPIv4.0].

If TLV 23 is not present in the MDD message, a CM MUST assume the CMTS only supports BPI+ V1.

7.1.4 Trust On First Use (TOFU)

With the introduction of multiple versions of BPI+, there is the potential risk for unauthorized downgrades from one version of BPI+ to another. Specifically, the CM is required to keep track of successful authentication configuration and use that information to prevent using older versions of BPI+ unless explicitly allowed to do so by the CMTS and only after a successful authentication. Specifically, the Trust On First Use (TOFU) approach provides protection against unauthorized downgrades by allowing CMs to trust the BPI+ parameters used during the first connection to a CMTS and then check that the same parameters (or better) are used in subsequent connections.

The primary element of TOFU is the set of Persistent Security Attribute (PSA) values, see Section 12.2 for details, that are stored on the CM. PSA values have the following characteristics:

1. The initial default PSA values allow a CM to authenticate with the CMTS by using any supported BPI+ versions. A CM where no previous PSA values are stored (e.g., after a factory reset) MUST allow for any version of BPI+ to be used during the authorization process.

2. After a successful authentication, the CM PSA values are updated to the used version of the protocol unless the CMTS indicates that earlier versions of the protocol are still allowed by including the Allowed-BPI-Versions Attribute TLV in the Authorization Reply message or by directly configuring CM PSA values via an authenticated management interface. For example, a CM that authenticates with BPI+ V2 can have its PSA values configured to still allow BPI+ V1 for future authentications in case BPI+ V2 is not available. This feature can be used when mixed deployment exists where BPI+ V2 has not been enabled across the entire network.
3. CM PSA values can be set (or reset) via authenticated management interfaces such as SSH.
4. The CMTS can set (or reset) PSA values only when BPI+ V2 is used.
5. CM PSA values can be used to restrict CMs to only authenticate with specific CMTSs (i.e., CMTS-pinning) and/or enable only specific BPI+ versions.
6. CM PSA values can be reset to the initial default values via a physical factory reset of the CM.

This combination of features enable the secure provisioning of CMs while still enabling the relocation of a CM onto a different CMTS that supports different BPI+ versions and/or present different values in their certificates or their Issuing CA's certificates (see Section 7.1.4.4).

The ability to reset the configuration of PSA values by CM factory reset enables the physical relocation of a CM or that CM to change service providers without requiring support or intervention from the operator.

When changes in PSA values are needed, the CMTS operator can use the following options to update the PSA values:

a) reset the version and/or certificate attributes stored in the PSA values by forcing a re-authentication of the CM before moving the CM to the new CMTS, or

b) reset the PSA values via a management interface (e.g., SNMP or SSH) after a successful authentication but still before moving the CM to the new CMTS. After the CM successfully authenticates to the new CMTS, new PSA values are sent to the CM to bind the CM to the new CMTS for subsequent authentications.

A CM MUST implement the TOFU mechanism as defined in this specification.

A CM MUST reject setting (or resetting) of PSA values when BPI+ V1 is used for authorization.

A CM MUST allow setting (or resetting) of PSA values when BPI+ V2 is used for authentication and authorization.

A CM MUST support factor reset where PSA values are cleared to factory default.

7.1.4.1 The Persistent Security Attribute Values

The CM's Persistent Security Attribute (PSA) values are used to persistently store security-related data across CM initializations and power cycles. PSA values are used to prevent unauthorized downgrades and provide additional network authentication controls, e.g., CMTS identity pinning.

In order to prevent unauthorized downgrades from one version of BPI+ to another, when the CM issues a new Authorization Request message, the CM first checks the values of the allowed BPI+ versions to select the latest supported version of BPI+ among the ones announced by the CMTS (see Section 7.1.3).

PSA values are also used to specify restrictions to the presented CMTS Certificate or CMTS CA certificate during authorization. In particular, when receiving either an Authorization Reply message or an Authorization Reject message, the CM checks the values configured in the `CMTS-Designation` Attributes against the CMTS Certificate or in the CMTS CA Certificate (e.g., set the accepted value(s) for the Organization (O) field in the certificate subject) and rejects the connection if the values do not match (see Section 7.2.2.38).

The CM MUST retain the values of the PSAs across CM initializations and power cycles.

The CM MUST set the PSA values to the default factory values when no previous configuration exists.

The CMTS MUST allow an operator to set specific values for the `Allowed-BPI-Versions` Attribute to set the minimum version of BPI+ the CM is allowed to use.

Table 6 - Persistent Security Attribute Values

Attribute	Contents	Default	See
Allowed-BPI-Versions	Allowed BPI+ versions for future authorizations	0	Section 7.2.2.36
CMTS-Designation	CMTS designation parameters	None	Section 7.2.2.38

7.1.4.2 PSA Value Reset

When the configured PSA values are not supported by the CMTS (e.g., the CM has been moved to a different location where BPI+ V2 is not enabled), the CM refuses to complete the authorization process until the PSA values are reset, possibly through a factory reset of the CM or by the operator resetting the PSA values through a management interface.

To allow PSA values to be cleared, the CM MUST support a method to reset the PSA values that requires physical access to the device to operate it (i.e., cannot be operated remotely).

For instance, the PSAs reset can be enabled as part of the CM factory reset process.

The CM MUST set the value of the PSAs to the default factory values when the reset mechanism is operated.

7.1.4.3 Enabling Support for Multiple BPI+ Versions

To allow for either planned downgrades or when BPI+ V2 is not enabled on every CMTS, operators need the possibility to configure the CM to enable BPI+ V1 for future authentications even when the CM successfully authenticates by using BPI+ V2.

The CMTS can use the `Allowed-BPI-Versions` in the Authorization Reply message to restrict which versions (can be more than one) of BPI+ can be enabled on the CM.

The CM MUST store the values carried in the `Allowed-BPI-Versions` Attribute in the Authorization Reply message in its PSA values only after a successful BPI+ V2 authentication and authorization, i.e., after the CM successfully verifies the signature over the message contents, validates the identity of the signing CMTS, verifies the CMTS meets the requirements of the current PSA values, and the CM is able to derive the Authorization Key as part of the authorization process (see Section 7.1.1.2, Authorization Messages Authentication Overview).

The CM MUST ignore the values carried in the `Allowed-BPI-Versions` Attribute in the Authorization Reply message when BPI+ V2 authorization is not successfully completed.

A CM MUST NOT issue authorization messages for BPI+ versions that are not enabled in the configured PSA values.

A CM MUST ignore authorization messages for BPI+ versions that are not explicitly included in the PSA values.

See Section 7.1.3 for more information on determining the BPI+ version.

7.1.4.4 CMTS Certificate Designation

When additional checks are required to make sure that the CM only connects to specific CMTSs, it is possible to configure the PSA values to require specific values for the CMTS certificate or the CMTS CA certificate.

The `CMTS-Designation` Attribute in the Authorization Reply message can be used to restrict the identities of which CMTSs the CM is allowed to authenticate with. The use of `CMTS-Designation` Attributes provides certificate pinning capabilities for both the CMTS Certificate and the CMTS CA Certificate.

For instance, to restrict the organization names that CMs will accept during network authentication, the CMTS sets the acceptable value of the Organization (O) field in the PSA values (e.g., “CableLabs”) that will be used in future authorizations.

A CMTS MUST allow the operator to configure the `CMTS-Designation` Attribute with specific values for either the elements of the CMTS Certificate or the CMTS CA Certificate.

A CMTS MUST allow the operator to configure the `CMTS-Designation` Attribute either globally or on per-CM basis.

A CM MUST verify that the values configured via the `CMTS-Designation` Attributes in the PSA values match the values in either the CMTS Certificate or in the CMTS CA Certificate (see Section 7.2.2.38, CMTS-Designation).

A CM MUST reject messages from a CMTS when either the CMTS Certificate or the CMTS CA Certificate parameters differ from the configured ones in the PSA values.

7.1.5 Selecting Cryptographic Suites

As part of their authorization exchange, the CM provides the CMTS with a list of supported cryptographic suites. The CMTS selects from this list a single suite to use with the CM's Primary SA. In the Authorization Reply, the CMTS includes a Primary SA descriptor that identifies the cryptographic suite selected by the CMTS.

The CMTS MUST reject the Authorization Request if none of the offered cryptographic suites is permitted by local policy.

The Authorization Reply may contain a list of Static SA descriptors; each Static SA descriptor identifies the cryptographic suite employed by that SA. The selection of a Static SA's cryptographic suite is independent of the requesting CM's cryptographic capabilities.

The CMTS MAY include in its Authorization Reply Static SA descriptors identifying cryptographic suites unsupported by the CM.

The CM MUST NOT start TEK state machines for SAs whose cryptographic suites the CM does not support.

7.1.6 Authorization State Machine

The Authorization Finite State Machine (FSM) contains six states and ten events. The Authorization FSM is presented below as a state flow diagram (Figure 8) and as a state transition matrix (Table 7).

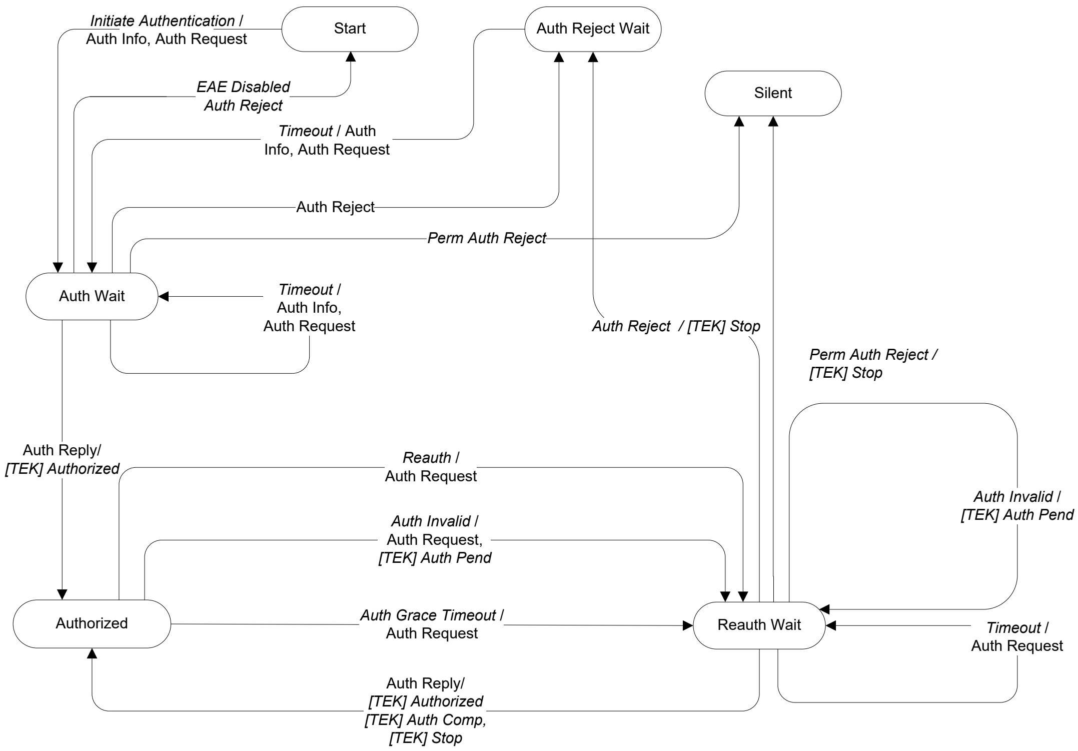
Initiate Authentication / Auth Info, Auth Request
Start
Auth Reject Wait
EAE Disabled Auth Reject
Silent
Timeout / Auth Info, Auth Request
Auth Reject
Perm Auth Reject
Auth Wait
Timeout / Auth Info, Auth Request
Auth Reject / [TEK] Stop
Perm Auth Reject / [TEK] Stop
Auth Reply/ [TEK] Authorized
Reauth / Auth Request
Auth Invalid / [TEK] Auth Pend
Auth Invalid / Auth Request, [TEK] Auth Pend
Authorized
Auth Grace Timeout / Auth Request
Reauth Wait
Timeout / Auth Request
Auth Reply/ [TEK] Authorized [TEK] Auth Comp, [TEK] Stop

Figure 8 - Authorization State Machine Flow Diagram

Table 7 - Authorization FSM Transition Matrix

State Event or Received Message	[Start]	[Auth Wait]	[Authorized]	[Reauth Wait]	[Auth Reject Wait]	[Silent]
{Initiate Authentication}	[Auth Wait]					
{Auth Reject}		[Auth Reject Wait]		[Auth Reject Wait]		
{Perm Auth Reject}		[Silent]		[Silent]		
{EAE Disabled Auth Reject}		[Start]				
{Auth Reply}		[Authorized]		[Authorized]		
{Timeout}		[Auth Wait]		[Reauth Wait]	[Auth Wait]	
{Auth Grace Timeout}			[Reauth Wait]			
{Auth Invalid}			[Reauth Wait]	[Reauth Wait]		
{Reauth}			[Reauth Wait]			
{Unsupported BPI+ Version}	[Silent]	[Start]				

The state flow diagram depicts the protocol messages transmitted and internal events generated for each of the machine's state transitions; however, the diagram does not indicate additional internal actions, such as the clearing or starting of timers that accompany the specific state transitions. Accompanying the state transition matrix is a detailed description of the specific actions accompanying each state transition.

The CM MUST use the text associated with the state transition matrix as the definitive specification of protocol actions associated with each state transition.

The following legend applies to Figure 8:

- States are represented by ovals.
- Events appear in italics.
- Messages appear in normal font.
- State transitions (i.e., the lines between states) are labeled in the manner of "<what causes the transition>/<messages and events triggered by the transition>." So "timeout/Auth Request" means that the state received a "timeout" event and sent an Auth Request message. If there are multiple comma-separated events or messages before the slash "/", any of them can cause the transition. If there are multiple events or messages listed after the slash, all the identified actions accompany the transition.

The Authorization state transition matrix, presented in Table 7, lists the states in the topmost row and the events in the left-most column. Cells within the matrix represent a specific combination of state and event, with the next state (the state to which the machine transitioned) displayed within the cell. For example, one cell represents the receipt of an Authorization Reply message when in the [Auth Wait] state. Within this cell is the name of the next state, [Authorized]. Thus, when a CM's Authorization state machine is in the [Auth Wait] state and an Authorization Reply message is received, the Authorization state machine will transition to the [Authorized] state. In conjunction with this state transition, several protocol actions are taken; these are described in Section 7.1.6.7.

A shaded cell within the state transition matrix (Table 7) implies that the specific event should not occur within that state.

If the event does occur, the CM MUST NOT transition to another state. For example, if an Authorization Reply message arrives when in the [Authorized] state, that message will not cause any state transition to occur.

The CM MAY, in response to such an improper event, log the event's occurrence, generate an SNMP event, or take some other vendor-defined action.

7.1.6.1 Brief Description of States

7.1.6.1.1 [Start]

This is the initial state of the FSM. No resources are assigned to or used by the FSM, all timers are off, and no processing is scheduled.

7.1.6.1.2 [Auth Wait]

The CM has received the {Initiate Authentication} event. In response to receiving the event, the CM has sent both an Authentication Information and an Authorize Request message and is waiting for the reply.

7.1.6.1.3 [Authorized]

The CM has received an Authorization Reply message that contains a list of valid SAIDs for this CM. The CM has a valid Authorization Key and the list of SAIDs. Transition into this state triggers the creation of one TEK FSM for each of the CM's privacy-enabled SAIDs.

7.1.6.1.4 [Reauth Wait]

The CM has an outstanding reauthorization request. The CM's current authorization is about to time out, or the CM has received an indication (an Authorization Invalid message) that its authorization is no longer valid. The CM has sent an Authorization Request message to the CMTS and is waiting for a response.

7.1.6.1.5 [Auth Reject Wait]

The CM received an Authorization Reject message in response to its last Authorization Request. The error code in the Authorization Reject indicated that the error was not permanent or that EAE is not disabled. In response to receiving this reject message, the CM sets a timer and has transitioned to the [Auth Reject Wait] state. The CM remains in this state until the timer expires.

7.1.6.1.6 [Silent]

The CM received an Authorization Reject message in response to its last Authorization Request and the error code indicated that the error was permanent or the CM and CMTS fail to properly authenticate. This triggers a transition to the [Silent] state.

In the [Silent] state, the CM MUST NOT pass CPE traffic.

In the [Silent] state, if the CM has a valid IP address, then the CM MUST respond to SNMP management requests arriving from the cable network.

The CMTS can send unencrypted data traffic to a CM on an SAID for which it has sent an Authorization Reject message, or the CMTS can block such traffic.

7.1.6.2 Brief Description of Messages

The corresponding message formats are specified in Section 7.2.

7.1.6.2.1 Authorization Request (Auth Request)

Request an Authorization Key and a list of authorized SAIDs. The Authorization Request is sent from the CM to the CMTS.

7.1.6.2.2 Authorization Reply (Auth Reply)

Receive an Authorization Key or a set of key exchange public parameters, and a list containing the Primary SAID and Static SAIDs. The Authorization Reply is sent the CMTS to the CM. The Authorization Key is encrypted with the CM's public key (BPI+ V1) or derived from the combination of the CM parameters and the CMTS parameters (BPI+ V2).

7.1.6.2.3 Authorization Reject (Auth Reject)

The attempt to authorize was rejected. The Authorization Reject is sent from the CMTS to the CM.

7.1.6.2.4 Authorization Invalid (Auth Invalid)

The CMTS can send an Authorization Invalid message to a client CM as

- an unsolicited indication or
- a response to a message received from that CM.

In either case, the Authorization Invalid message instructs the receiving CM to reauthorize with its CMTS.

The CMTS MUST respond to a Key Request with an Authorization Invalid message if

1. There is no valid Authorization Key associated with the CM or
2. Verification of the Key Request's keyed message digest (in the HMAC-Digest Attribute) failed.

7.1.6.2.5 Authentication Information (Auth Info)

The Authentication Information message contains the intermediate CA certificate (see Section 7.2.1.9) issued by the Root CA that issued the CM's certificate. The Auth Information message is sent by the CM to the CMTS.

The CMTS MUST first use the out-of-band configuration information to obtain the intermediate CA certificate (see Section 13.3.2, Certificate Validation).

If the CMTS does not learn the intermediate CA certificate from the out-of-band configuration information, then the CMTS MUST use the intermediate CA certificate from the Auth Info message sent by the CM.

7.1.6.2.6 Authentication Status Info (Auth Status Info)

The Authentication Status Info message is only used in BPI+ V2 and carries the status code for the results of the CM’s authentication of the CMTS credentials to the CMTS itself. The message provides an indication of error(s) that have occurred on the CM when validating the CMTS’ message.

7.1.6.3 Brief Description of Events

7.1.6.3.1 {Initiate Authentication}

The CM sends an {Initiate Authentication} event to the Authorization FSM upon completing CMTS registration or, if EAE is enabled, when it completes ranging. If the configuration file contains a BPI+ enable setting (see Annex A.1.1) and the CM has not yet enabled BPI+, then the {Initiate Authentication} event causes the CM to begin the process of obtaining its Authorization Key and transitions to the [Auth Wait] state.

7.1.6.3.2 {Timeout}

This event indicates that a retransmission or wait timer timed out. Generally, a request is resent.

7.1.6.3.3 {Auth Grace Timeout}

This event indicates that the Authorization Grace timer fired. This timer fires a configurable duration (the Authorization Grace Time) before the current authorization expires, signaling the CM to reauthorize before its authorization actually expires. Once the configuration file has been received by the CM, the Authorization Grace Time is obtained from a value in that file; prior to that time, the value of the Authorization Grace Time MUST be the default defined in Annex A.

7.1.6.3.4 {Reauth}

This event is generated in response to an SNMP SET [DOCSIS CM-OSSIv4.0] that is intended to trigger a reauthorization cycle because the CM's set of authorized Static SAIDs may have changed.

7.1.6.3.5 {Auth Invalid}

This event is generated when there is a failure authenticating a Key Reply, Key Reject, or TEK Invalid message or when the CM receives an Authorization Invalid message. A CMTS responds to a Key Request with an Authorization Invalid message if verification of the request's message authentication code fails. Generation of an {Auth Invalid} event indicates that the CMTS and the CM have lost Authorization Key synchronization.

A CMTS MAY send an unsolicited Authorization Invalid message to a CM, forcing an {Auth Invalid} event.

7.1.6.3.6 {Perm Auth Reject}

This event indicates receipt of an Authorization Reject message containing either error code 6 or error code 11. When a CM receives an Authorization Reject containing one of these error codes, the Authorization State machine moves to the [Silent] state.

7.1.6.3.7 {Auth Reject}

This event indicates that the CM has received an Authorization Reject in response to an Authorization Request and that the error code in the Authorization Reject had some value other than 6 or 10. The CM's Authorization state machine will set a wait timer and transition into the [Auth Reject Wait] state. The CM remains in this state until the timer expires, at which time it will re-attempt authorization.

7.1.6.3.8 {Unsupported BPI-Version}

This event indicates that the BPI+ version the CM is attempting to use is not supported by the CMTS. This condition can be detected:

1. by the CM when comparing the supported and enabled BPI+ version(s) to the version(s) signaled by the CMTS and finding no compatible version, or
2. by the CMTS receiving an Authorization Request message with an incompatible BPI+ version and issuing an Authorization Reject message with error code 11.

7.1.6.3.9 {EAE Disabled Auth Reject}

This event indicates that the CM has received an Authorization Reject message containing error code 10 in response to an Authorization Request that was sent to the CMTS as part of EAE. Authorization Reject error code 10 messages are sent to CMs that have been specifically configured in the CMTS to have EAE disabled. When receiving this message, the CM's Authorization state machine will transition to the [Start] state.

7.1.6.4 Events Sent to TEK State Machine

The following events are sent by the Authorization state machine to a TEK state machine.

7.1.6.4.1 {TEK Stop}

Sent by the Authorization FSM to a TEK FSM that is not in the [Start] state to terminate the TEK FSM and remove the corresponding SAID's keying material.

7.1.6.4.2 {TEK Authorized}

Sent by the Authorization FSM to a TEK FSM that is in the [Start] state.

7.1.6.4.3 {Auth Pend}

This message is sent by the Authorization FSM to a TEK FSM to place that TEK FSM into a wait state until the Authorization FSM can complete a reauthorization operation.

7.1.6.4.4 {Auth Comp}

Sent by the Authorization FSM to a TEK FSM in the [Op Reauth Wait] or [Rekey Reauth Wait] states to clear the wait state begun by an {Auth Pend} event.

7.1.6.5 Brief Description of Timing Parameters

All configuration parameter values are contained in the configuration file (see Annex A).

7.1.6.5.1 Authorize Wait Timeout (Auth Wait Timeout)

This is the timeout period between sending Authorization Request messages when in the [Auth Wait] state.

7.1.6.5.2 Reauthorize Wait Timeout (Reauth Wait Timeout)

This is the timeout period between sending Authorization Request messages when in the [Reauth Wait] state.

7.1.6.5.3 Authorization Grace Time (Auth Grace Timeout)

Amount of time before authorization is scheduled to expire that the CM starts the process of reauthorization.

7.1.6.5.4 Authorize Reject Wait Timeout (Auth Reject Wait Timeout)

Amount of time that a CM's Authorization FSM remains in the [Auth Reject Wait] state before re-attempting authorization.

7.1.6.6 Timers

7.1.6.6.1 Authorization Request

Used when awaiting a response to Authorization Requests.

7.1.6.6.2 Authorization Reject

Used after receipt of an Authorization Reject.

7.1.6.6.3 Authorization Grace

Used when determining when to reauthorize.

7.1.6.7 Actions

The CM MUST take the following actions in association with state transitions.

[Start] + {Initiate Authentication} → [Auth Wait]:

- Send Authentication Information message to CMTS;
- Send Authorization Request message to CMTS;
- Set Authorization Request timer to Authorize Wait Timeout.

[Start] + {Unsupported BPI+ Version} → [Silent]:

- Disable all forwarding of CPE traffic.
- Send the Authorization Status Info to the CMTS indicating the specific error.

[Auth Wait] + {Auth Reject} → [Auth Reject Wait]:

- Clear Authorization Request timer;
- Set Authorization Reject timer to Authorize Reject Wait Timeout.

[Reauth Wait] + {Auth Reject} → [Auth Reject Wait]:

- Clear Authorization Request timer;
- Generate TEK FSM {Stop} events for all active TEK state machines;
- Set Authorization Reject timer to Authorize Reject Wait Timeout.

[Auth Wait] + {Perm Auth Reject} → [Silent]:

- Clear Authorization Request timer;
- Disable all forwarding of CPE traffic.

[Auth Wait] + {EAE Disabled Auth Reject} → [Start]:

- Clear Authorization Request timer.

[Auth Wait] + {Unsupported BPI+ Version} → [Start]:

- Clear Authorization Request timers;
- Send the Authorization Status Info to the CMTS indicating the specific error.

[Reauth Wait] + {Perm Auth Reject} → [Silent]:

- Clear Authorization Request timer;
- Generate TEK FSM {Stop} events for all active TEK state machines;
- Disable all forwarding of CPE traffic.

[Auth Wait] + {Auth Reply} → [Authorized]:

- (BPI+ V2 only) Validate the authentication information (signature) of the received Authorization and send an Auth Status Info with the results of the authentication back to the CMTS. If authentication fails, the CM discards the received { Auth Reply } message resulting in no state changes;
- Clear Authorization Request timer;
- Decrypt (BPI+ V1) or derive (BPI+ V2) and record the Authorization Key delivered with Authorization Reply;
- Start TEK FSMs for all SAIDs listed in Authorization Reply and any pending Dynamic SAIDs received in Registration Response (provided that the CM supports the cryptographic suite that is associated with an SAID) and issue a TEK FSM {Authorized} event for each of the new TEK FSMs;
- Set the Authorization Grace timer to fire "Authorization Grace Time" seconds prior to the supplied Authorization Key's scheduled expiration.

[Reauth Wait] + {Auth Reply} → [Authorized]:

- (BPI+ V2 only) Validate the authentication information (signature) of the received Authorization and send an Auth Status Info with the results of the authentication back to the CMTS. If authentication fails, the CM discards the received { Auth Reply } message resulting in no state changes;
- Clear Authorization Request timer;
- Decrypt (BPI+ V1) or derive (BPI+ V2) and record the Authorization Key delivered with Authorization Reply;
- Start TEK FSMs for any newly authorized SAIDs listed in Authorization Reply (provided that the CM supports the cryptographic suite that is associated with the new SAID) and issue TEK FSM {Authorized} events for each of the new TEK FSMs;
- Generate TEK FSM {Authorization Complete} events for any currently active TEK FSMs whose corresponding SAIDs were listed in Authorization Reply;
- Generate TEK FSM {Stop} events for any currently active TEK FSMs whose corresponding Primary or Static SAIDs were not listed in Authorization Reply;
- Set the Authorization Grace timer to fire "Authorization Grace Time" seconds prior to the supplied Authorization Key's scheduled expiration.

[Auth Wait] + {Timeout} → [Auth Wait]:

- Send Authentication Information message to CMTS;
- Send Authorization Request message to CMTS;
- Set Authorization Request timer to Authorize Wait Timeout.

[Reauth Wait] + {Timeout} → [Reauth Wait]:

- Send Authorization Request message to CMTS;
- Set Authorization Request timer to Reauthorize Wait Timeout.

[Auth Reject Wait] + {Timeout} → [Auth Wait]:

- Send Authentication Information message to CMTS;
- Send Authorization Request message to CMTS;
- Set Authorization Request timer to Authorize Wait Timeout.

[Authorized] + {Auth Grace Timeout} → [Reauth Wait]:

- Send Authorization Request message to CMTS;
- Set Authorization Request timer to Reauthorize Wait Timeout.

[Authorized] + {Auth Invalid} → [Reauth Wait]:

- Clear Authorization Grace timer;
- Send Authorization Request message to CMTS;
- Set Authorization Request timer to Reauthorize Wait Timeout;
- If the Authorization Invalid event is associated with a particular TEK FSM, generate a TEK FSM {Authorization Pending} event for the TEK state machine responsible for the Authorization Invalid event (i.e., the TEK FSM that either generated the event or sent the Key Request message to which the CMTS responded with an Authorization Invalid message).

[Reauth Wait] + {Auth Invalid} → [Reauth Wait]:

- If the Authorization Invalid event is associated with a particular TEK FSM, generate a TEK FSM {Authorization Pending} event for the TEK state machine responsible for the Authorization Invalid event (i.e., the TEK FSM that either generated the event or sent the Key Request message to which the CMTS responded with an Authorization Invalid message).

[Authorized] + {Reauth} → [Reauth Wait]:

- Clear Authorization Grace timer;
- Send Authorization Request message to CMTS;
- Set Authorization Request timer to Reauthorize Wait Timeout.

7.1.7 TEK State Machine

The TEK state machine contains six states and nine events. The TEK state machine is presented as a state flow diagram in Figure 9 and a state transition matrix in Table 8.

The CM MUST use the state transition matrix, together with the required actions in Section 7.1.7.6, Actions, as the definitive specification of protocol actions associated with each state transition.

Shaded states in Figure 9 ({Operational}, {Rekey Wait}, and {Rekey Reauthorize Wait}) indicate that the CM holds valid keying material, so encrypted traffic can be passed.

The Authorization state machine starts an independent TEK state machine for each of its authorized SAIDs.

The CMTS maintains two active TEKs per SAID. The CMTS includes in its Key Replies both of these TEKs, along with their remaining lifetimes. The CMTS encrypts downstream traffic with the older of its two TEKs and decrypts upstream traffic with either the older or newer TEK, depending on which of the two keys the CM used. The CM encrypts upstream traffic with the newer of its two TEKs and decrypts downstream traffic with either the older or newer key, depending on which of the two keys the CMTS used. See Sections 10.1 and 10.2 for details on CMTS and CM key usage requirements, respectively.

Through operation of a TEK state machine, the CM attempts to keep its copies of an SAID's TEKs, synchronized with those of its CMTS. A TEK state machine issues Key Requests to refresh copies of its SAID's keying material after the scheduled expiration time of the older of its two TEKs and before the expiration of its newer TEK. To accommodate CM/CMTS clock skew and other system processing and transmission delays, the CM schedules its Key Requests a configurable number of seconds (the TEK Grace Time) before the newer TEK expires.

When it receives a Key Reply, the CM MUST immediately begin to use the TEK Parameters from the TEKs contained in the Key Reply Message.

Figure 9 illustrates the CM's scheduling of key refreshes in conjunction with its management of an SA's active TEKs.

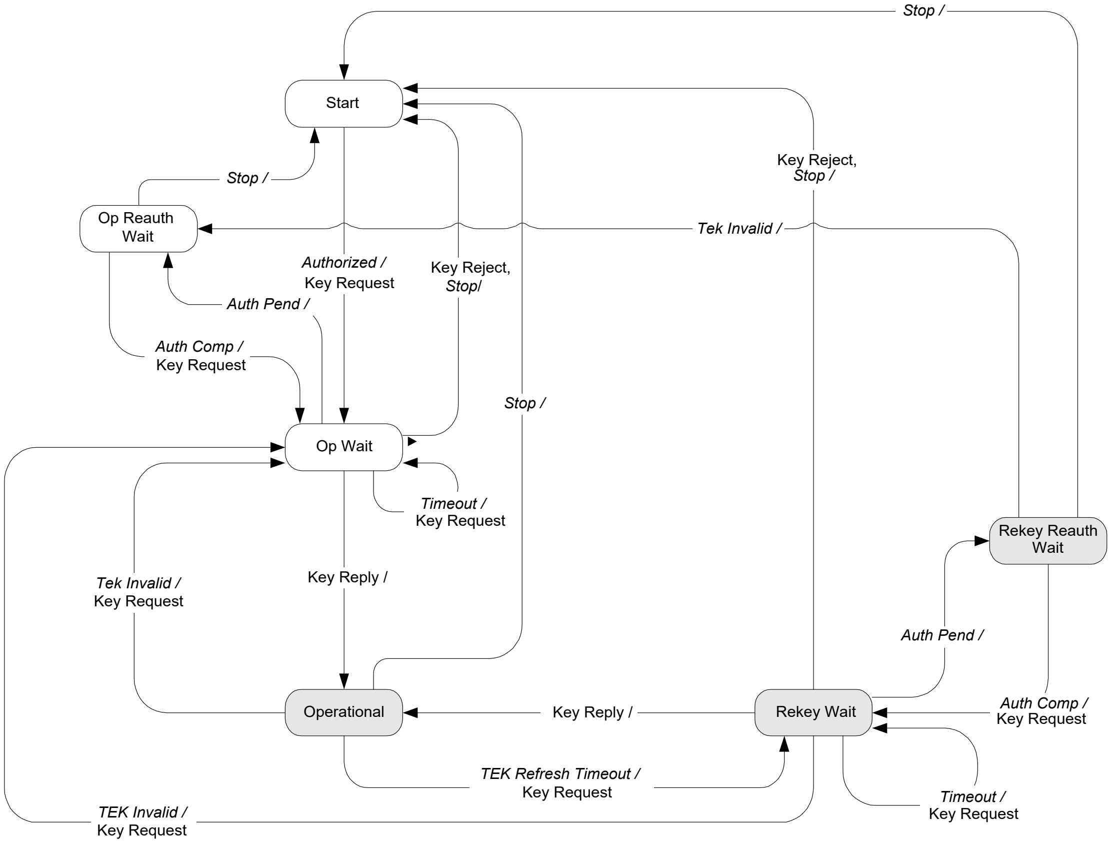

Figure 9 - TEK State Machine Flow Diagram

Table 8 - TEK FSM State Transition Matrix

State Event or Rcvd Message	[Start]	[Op Wait]	[Op Reauth Wait]	[Op]	[Rekey Wait]	[Rekey Reauth Wait]
{Stop}		[Start]	[Start]	[Start]	[Start]	[Start]
{Authorized}	[Op Wait]					

State Event or Rcvd Message	[Start]	[Op Wait]	[Op Reauth Wait]	[Op]	[Rekey Wait]	[Rekey Reauth Wait]
{Auth Pend}		[Op Reauth Wait]			[Rekey Reauth Wait]	
{Auth Comp}			[Op Wait]			[Rekey Wait]
{TEK Invalid}				[Op Wait]	[Op Wait]	[Op Reauth Wait]
{Timeout}		[Op Wait]			[Rekey Wait]	
{TEK Refresh Timeout}				[Rekey Wait]		
{Key Reply}		[Op]			[Op]	
{Key Reject}		[Start]			[Start]	

7.1.7.1 Brief Description of States

7.1.7.1.1 [Start]

This is the initial state of the FSM. No resources are assigned to or used by the FSM, all timers are off, and no processing is scheduled.

7.1.7.1.2 [Op Wait]

The TEK state machine has sent its initial Key Request for its SAID's keying material (TEK and CBC IV) and is waiting for a reply from the CMTS.

7.1.7.1.3 [Op Reauth Wait]

The Authorization state machine is in a reauthorization cycle, and the CM does not have valid keying material for this SAID.

7.1.7.1.4 [Op]

The CM has valid keying material for the associated SAID.

7.1.7.1.5 [Rekey Wait]

The TEK Refresh Timer has expired, and the CM has requested a key update for this SAID to replace the older of the two TEKs.

7.1.7.1.6 [Rekey Reauth Wait]

The CM has valid traffic keying material for this SAID and has an outstanding request for the latest keying material, and the Authorization state machine has initiated a reauthorization cycle.

7.1.7.2 Brief Description of Messages

The message formats are defined in detail in Section 7.2. All messages contain a keyed message digest for which the key is derived from the Authorization Key (see Section 11.4).

7.1.7.2.1 Key Request

Request a TEK for this SAID. The Key Request is sent by the CM to the CMTS.

7.1.7.2.2 Key Reply

Response from the CMTS carrying the two active sets of traffic keying material for this SAID. It includes the SAID's TEKs, 3DES encrypted with a KEK derived from the Authorization Key (see Section 11.4).

7.1.7.2.3 Key Reject

If the SAID in a Key Request message is invalid, then the CMTS MUST respond with a Key Reject message.

7.1.7.2.4 TEK Invalid

If the TEK used to encrypt an upstream PDU is invalid, then the CMTS MUST respond with a TEK Invalid message.

7.1.7.3 Brief Description of Events

7.1.7.3.1 {Stop}

This event is sent by the Authorization FSM to a TEK FSM not in the [Start] state to terminate the TEK FSM and remove the corresponding SAID's keying material (see Section 7.1.6.4.1).

7.1.7.3.2 {Authorized}

This event is sent by the Authorization FSM to a TEK FSM in the [Start] state to notify the TEK FSM of successful authorization (see Section 7.1.6.4.2).

7.1.7.3.3 {Auth Pend}

This event is sent by the Authorization FSM to a TEK FSM to place the TEK FSM in a wait state while the Authorization FSM completes reauthorization (see Section 7.1.6.4.3).

7.1.7.3.4 {Auth Comp}

This event is sent by the Authorization FSM to a TEK FSM in the [Op Reauth Wait] or [Rekey Reauth Wait] states to clear the wait state begun by a prior {Auth Pend} event (see Section 7.1.6.4.4).

7.1.7.3.5 {TEK Invalid}

This event can be triggered either by a CM's data packet decryption logic or by the receipt of a TEK Invalid message from the CMTS.

A CM triggers a {TEK Invalid} event when it recognizes loss of TEK key synchronization between itself and the encrypting CMTS; i.e., an SAID's TEK key sequence number, contained within the received downstream packet's Baseline Privacy EH element, is out of the CM's range of known sequence numbers for that SAID.

A CMTS sends the CM a TEK Invalid message, triggering a {TEK Invalid} event if the CMTS recognizes a loss of TEK key synchronization between itself and the CM.

7.1.7.3.6 {Timeout}

This specifies the period of the Retry Timer timeout. Generally, the original request is retransmitted.

7.1.7.3.7 {TEK Refresh Timeout}

This is the period defined for the TEK Refresh Timer. This event instructs the TEK state machine to issue a new Key Request in order to refresh its keying material. The refresh timer can be configured for a specific length of time (the TEK Grace Time) before the expiration of the newer TEK currently held by the CM elapses.

7.1.7.4 Brief Description of Timing Parameters

All configuration parameter values are contained in the configuration file (see Annex A).

7.1.7.4.1 Operational Wait Timeout

This value specifies the time period between consecutive Key Request messages whenever the state machine is in the [Op Wait] state.

7.1.7.4.2 Rekey Wait Timeout

The timeout period between the sending of Key Request messages when in the [Rekey Wait] state.

7.1.7.4.3 TEK Grace Time

The time interval, in seconds, prior to the expiration of a TEK at which the CM starts re-keying for a new TEK.

7.1.7.5 Timers

7.1.7.5.1 Key Request Retry

Used when the response to a Key Request was not received by the CM.

7.1.7.5.2 TEK Refresh

Used to initiate a request for keys.

7.1.7.6 Actions

The CM MUST take the following actions in association with state transitions:

[Op Wait] + {Stop} → [Start]:

- Clear Key Request Retry timer;
- Terminate TEK FSM.

[Op Reauth Wait] + {Stop} → [Start]:

- Terminate TEK FSM.

[Op] + {Stop} → [Start]:

- Clear TEK Refresh timer;
- Terminate TEK FSM;
- Remove SAID keying material.

[Rekey Wait] + {Stop} → [Start]:

- Clear Key Request Retry timer;
- Terminate TEK FSM;
- Remove SAID keying material.

[Rekey Reauth Wait] + {Stop} → [Start]:

- Terminate TEK FSM;
- Remove SAID keying material.

[Start] + {Authorized} → [Op Wait]:

- Send Key Request Message to CMTS;
- Set Key Request Retry timer to the value of Operational Wait Timeout.

[Op Wait] + {Auth Pend} → [Op Reauth Wait]:

- Clear Key Request Retry timer.

[Rekey Wait] + {Auth Pend} → [Rekey Reauth Wait]:

- Clear Key Request Retry timer.

[Op Reauth Wait] + {Auth Comp} → [Op Wait]:

- Send Key Request message to CMTS;
- Set Key Request Retry timer to Operational Wait Timeout.

[Rekey Reauth Wait] + {Auth Comp} → [Rekey Wait]:

- Send Key Request message to CMTS;
- Set Key Request Retry timer to Rekey Wait Timeout.

[Op] + {TEK Invalid} → [Op Wait]:

- Clear TEK Refresh timer;
- Send Key Request message to CMTS;
- Set Key Request Retry timer to Operational Wait Timeout;
- Remove SAID keying material.

[Rekey Wait] + {TEK Invalid} → [Op Wait]:

- Clear Key Request Retry timer;
- Send Key Request message to CMTS;
- Set Key Request Retry timer to Operational Wait Timeout;
- Remove SAID keying material.

[Rekey Reauth Wait] + {TEK Invalid} → [Op Reauth Wait]:

- Remove SAID keying material.

[Op Wait] + {Timeout)} → [Op Wait]:

- Send Key Request message to CMTS;
- Set Key Request Retry timer to Operational Wait Timeout.

[Rekey Wait] + {Timeout} → [Rekey Wait]:

- Send Key Request message to CMTS;
- Set Key Request Retry timer to Rekey Wait Timeout.

[Op] + {TEK Refresh Timeout} → [Rekey Wait]:

- Send Key Request message to CMTS;
- Set Key Request Retry timer to Rekey Wait Timeout.

[Op Wait] + {Key Reply} → [Op] (Key Reply passed message authentication):

- Clear Key Request Retry timer;
- Process contents of Key Reply message and incorporate new keying material;
- Set the TEK Refresh timer to fire "TEK Grace Time" seconds prior to the key's scheduled expiration.

[Rekey Wait] + {Key Reply} → [Op] (Key RejectReply passed message authentication):

- Clear Key Request Retry timer;
- Process contents of Key Reply message and incorporate new keying material;
- Set the TEK Refresh timer to fire "TEK Grace Time" seconds prior to the key's scheduled expiration.

[Op Wait] + {Key Reject} → [Start] (Key RejectReply passed message authentication):

- Clear Key Request Retry timer;
- Terminate TEK FSM.

[Rekey Wait] + {Key Reject} → [Start]:

- Clear Key Request Retry timer;
- Terminate TEK FSM;
- Remove SAID keying material.

7.2 Key Management Message Formats

BPKM employs four MAC Management Message types: version 1 BPKM-REQ and BPKM-RSP for BPI+ V1 MMMs, and version 5 BPKM-REQ and BPKM-RSP for BPI+ V2 MMMs.

The CMTS MUST support the BPKM formats as defined in this section (Section 7.2, Key Management Message Formats).

The CM MUST support the BPKM formats as defined in this section (Section 7.2, Key Management Message Formats).

[DOCSIS MULPIv4.0] defines the specific type values assigned to these messages (see also Table 9).

All messages in this section carry the same set of Attributes across both BPI+ V1 and BPI+ V2, unless explicitly noted. Attributes that are marked as "Common" are present in both versions of BPI+. Attributes marked for a specific version of BPI+ are present only when that specific version of BPI+ is used.

Table 9 - Baseline Privacy Key Management MAC Messages

Type Value	Message Name	Message Description
See [DOCSIS MULPIv4.0]	BPKM-REQ	Baseline Privacy Key Management Request for BPI+V1 (CM → CMTS)
See [DOCSIS MULPIv4.0]	BPKM-RSP	Baseline Privacy Key Management Response for BPI+V1 (CMTS → CM)
See [DOCSIS MULPIv4.0]	BPKM-REQ	Baseline Privacy Key Management Request for BPI+V2 (CM → CMTS)
See [DOCSIS MULPIv4.0]	BPKM-RSP	Baseline Privacy Key Management Response for BPI+V2 (CMTS → CM)

7.2.1 Packet Formats

This specification supports two versions of the authentication protocols: BPI+ V1 that is shared across DOCSIS 3.1 and 4.0 environments, and BPI+ V2 that is specific to DOCSIS 4.0 technology.

BPKM messages for BPI+ V1 use version 1 MAC Management Message types 12 and 13 as described in [DOCSIS MULPIv4.0]. BPKM MAC Management Message fragmentation support is not needed for BPI+ V1.

When negotiating BPI+ V1, the CM MUST encapsulate one (1) authorization message (e.g., Auth-Request) in the payload field of a single MAC management message (no fragmentation allowed).

When negotiating BPI+ V1, the CMTS MUST encapsulate one (1) authorization message (e.g., Auth-Reply) in the payload field of a single MAC management message (no fragmentation allowed).

BPKM messages for BPI+ V2 use version 5 MAC Management Message types 69 and 70 as described in [DOCSIS MULPIv4.0]. BPKM MAC Management Message fragmentation is supported for BPI+ V2.

In the rest of this specification, the term BPKM is used to refer to all supported MAC Management Message types for authorization messages, i.e., both BPI+ V1 type 12 and 13 MMMs and BPI+ V2 type 69 and 70 MMMs.

A summary of the BPKM message format follows.

The CMTS MUST transmit fields and their respective contents from left to right.

The CM MUST transmit fields and their respective contents from left to right.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|     Code      |  Identifier   |            Length             |
+---------------+---------------+---------------+---------------+
|   Attributes ...
+---------------+----
```

`Code`

The `Code` field is one octet and identifies the type of BPKM packet. The CM SHOULD silently discard a packet received with an invalid `Code` field.

BPKM Codes (decimal) are assigned as shown in Table 10.

Table 10 - Baseline Privacy Key Management Message Codes

Code	BPKM Message Type	MAC Management Message Name
0-3	Reserved	-
4	Auth Request	BPKM-REQ (Version 1 and Version 5 MMM)
5	Auth Reply	BPKM-RSP (Version 1 and Version 5 MMM)
6	Auth Reject	BPKM-RSP (Version 1 and Version 5 MMM)
7	Key Request	BPKM-REQ (Version 1 and Version 5 MMM)
8	Key Reply	BPKM-RSP (Version 1 and Version 5 MMM)
9	Key Reject	BPKM-RSP (Version 1 and Version 5 MMM)
10	Auth Invalid	BPKM-RSP (Version 1 and Version 5 MMM)
11	TEK Invalid	BPKM-RSP (Version 1 and Version 5 MMM)
12	Auth Info	BPKM-REQ (Version 1 and Version 5 MMM)
13	Map Request	BPKM-REQ (Version 1 and Version 5 MMM)
14	Map Reply	BPKM-RSP (Version 1 and Version 5 MMM)
15	Map Reject	BPKM-RSP (Version 1 and Version 5 MMM)
16	Auth Status Info	BPMK-REQ (Version 5 MMM)
17-255	Reserved	-

`Identifier`

The `Identifier` field is one octet. A CM uses this field to match a CMTS's responses to the CM's requests.

The CM MUST change the value of the `Identifier` field whenever it issues a new BPKM message.

A "new" message is an Authorization Request, Key Request, or SA Map Request that is not a retransmission sent in response to a {Timeout} event.

For retransmissions, the CM MUST keep the `Identifier` field unchanged from the value in the message being retransmitted.

The `Identifier` field in Authentication Information messages, which are informative and do not affect any response messaging, may be set to any value.

The CMTS MUST set the `Identifier` field of a BPKM response message to exactly match the `Identifier` field of the BPKM Request message to which the CMTS is responding.

The CMTS MUST set the `Identifier` field in TEK Invalid messages, which are not sent in response to BPKM requests, to zero.

The CMTS MUST set the `Identifier` field in unsolicited Authorization Invalid messages to zero.

On receipt of a BPKM response message, the CM MUST associate the message with a particular state machine (the Authorization state machine in the case of Authorization Replies, Authorization Rejects, and Authorization Invalids; a particular TEK state machine in the case of Key Replies, Key Rejects, and TEK Invalids, using the

SAID attribute in the BPKM response message; a particular SA Mapping state machine in the case of SA Map Replies and SA Map Rejects[1]).

The CM MAY keep track of the `Identifier` of a pending Authorization Request. The CM MAY silently discard Authorization Replies and Authorization Rejects whose `Identifier` fields do not match those of pending requests.

The CM MAY keep track of the `Identifier` of a pending Key Request. The CM MAY silently discard Key Replies and Key Rejects whose `Identifier` fields do not match those of pending requests.

The CM MAY keep track of the `Identifier` of a pending SA Map Request. The CM MAY silently discard SA Map Replies and SA Map Rejects whose `Identifier` fields do not match those of pending requests.

`Length`

[DOCSIS MULPIv4.0] defines Version 5 BPKM MAC Management Messages that can be up to 2000 bytes in length which also lengthens the maximum size of a DOCSIS MAC frame, after accounting for the DOCSIS MMM header including the maximum permitted size of the extended header. The Version 5 MMMs can be multipart, that is, fragmented across more than one DOCSIS MAC frame.

The `Length` field is two octets. It indicates the length of the Attribute fields, in octets. The value of the `Length` field does not include the length of the `Code`, `Identifier`, and `Length` fields.

If the packet contains more octets than indicated by the value of the `Length` field, then the CMTS MUST ignore the additional octets.

If the packet is shorter than indicated by the value of the `Length` field, then the CMTS SHOULD silently discard it.

If the packet contains more octets than indicated by the value of the `Length` field, then the CM MUST ignore the additional octets.

If the packet is shorter than indicated by the value of the `Length` field, then the CM SHOULD silently discard it.

When negotiating BPI+ V1, the CMTS MUST set the value of the `Length` field to fit a single MAC frame maximum size.

When negotiating BPI+ V1, the CM MUST set the value of the `Length` field to fit a single MAC frame maximum size.

When negotiating BPI+ V2, the CMTS MUST set the value of the Length field to be in the range supported by MAC Management Message payload maximum size (can be fragmented) supported by the Version 5 BPKM-RSP message.

When negotiating BPI+ V2, the CM MUST set the value of the Length field to be in the range supported by MAC Management Message payload maximum size (can be fragmented) supported by the Version 5 BPKM-REQ message.

`Attributes`

BPKM `Attributes` carry the specific authentication, authorization, and key-management data exchanged between client and server. Each BPKM packet type has its own set of required and optional `Attribute` fields. Unless explicitly stated, there are no requirements on the ordering of `Attribute` fields in a BPKM message.

The position of the end of the `Attribute` fields is calculated from the value of the `Length` field.

`Attribute` fields are type/length/value (TLV) encoded:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+----------------+
|    Type       |           Length              | Value ...
```

[1] The SA mapping mechanism was used in earlier versions of DOCSIS technology to support Dynamic SAs (see Annex C).

The BPKM MAC frame format is described in Section 6, and the BPKM packet format is described in Section 7.2.1. The descriptions below list the BPKM `Attributes` contained in each BPKM message type. The `Attribute` fields themselves are described in Section 7.2.2.

The CMTS MUST ignore unknown attributes on receipt.

The CM MUST ignore unknown attributes on receipt.

The CMTS MUST discard all messages that do not contain all the required attributes with valid values.

The CM MUST discard all messages that do not contain all the required attributes with valid values.

7.2.1.1 Authorization Request (Auth Request)

Code: 4

The form of Authorization Request sent by the CM is based upon the version of BPI+ indicated by the CMTS (see Section 7.1.3).

If the CMTS indicates support for BPI+ V2, the CM MUST issue a BPI+ V2 Authorization Request.

If the CM’s Persistent Security Attribute values indicate support for a version of BPI+ that’s not supported by the CMTS (see Section 12.2, Persistent Security Attribute Values), then the CM MUST NOT issue an Authorization Request.

In this case, the CM MUST both issue an Authorization Status Info message instead (see Section 7.2.1.13, Authentication Status Info (Auth Status Info)) and enter the {Silent} state (see Section 7.1.6, Authorization State Machine).

7.2.1.1.1 Common Authorization Request Attributes

Attribute List:

Table 11 - Common Authorization Request Attributes

Attribute	Contents	See
Security-Capabilities	Describes the CM's security capabilities	Section 7.2.2.19
SAID	Initialization SAID or CM's Primary SAID	Section 7.2.2.12

The `Security-Capabilities` attribute is a compound attribute describing the CM's security capabilities: supported data encryption algorithms, data authentication algorithms, and versions of the DOCSIS Security protocol.

The `SAID` attribute contains the value of an SAID. The SAID attribute contains the Initialization SAID (whose value is zero; see Section 7.1.1) if a CM is attempting initial Authorization and a CM's Primary SAID if a CM is attempting reauthorization.

7.2.1.1.2 BPI+ V1 Authorization Request Attributes

In addition to the Common Attributes, BPI+ V1 supports the attributes described in this section.

Attribute List:

Table 12 - BPI+ V1 Authorization Request Attributes

Attribute	Contents	See
CM-Identification	Contains information used to identify CM to CMTS	Section 7.2.2.5
CM-Certificate	Contains the CM's Device certificate	Section 7.2.2.18

The `CM-Identification` attribute contains data that identifies the CM to the CMTS. The public key in the RSA-Public-Key sub-attribute of the CM-Identification attribute is identical to the public key in the CM-Certificate attribute.

The `CM-Certificate` attribute contains a CM Device Certificate. The CM Device Certificate binds the CM's identifying information to its RSA public key. The certificate is signed by the CableLabs Device CA, and that signature can be verified by a CMTS that knows the corresponding public key.

7.2.1.1.3 BPI+ V2 Authorization Request Attributes

In addition to the Common Attributes, BPI+ V2 supports the attributes described in this section.

Attribute List:

Table 13 - BPI+ V2 Authorization Request Attributes

Attribute	Contents	See
BPI-Version	BPI+ version number	Section 7.2.2.22
Serial-Number	CM's serial number	Section 7.2.2.1
Manufacturer-ID	CM's manufacturer's ID	Section 7.2.2.2
CM-Nonce	A randomly generated NONCE value that is provided by the CM	Section 7.2.2.33
Key-Exchange-share	A key exchange share	Section 7.2.2.35
Message-Signature	A message signature attribute	Section 7.2.2.34

The `BPI-Version` attribute carries the version of BPI+.

The `Serial-Number` attribute contains the serial number identifier assigned by the manufacturer to a CM.

The `Manufacturer-ID` attribute provide the identifier of the CM's manufacturer.

The `CM-Nonce` attribute carries the NONCE value that ties the message to the corresponding request.

The `Key-Exchange-share` attribute contains one key share to be used to derive the BPI+ V2 authorization key.

The `Message-Signature` attribute contains a DER-encoded CMS signature over the header and attributes in a BPKM message.

7.2.1.2 Authorization Reply (Auth Reply)

Code 5:

Sent by the CMTS to a client CM in response to an Authorization Request, the Authorization Reply message contains authenticated ECDHE parameters (BPI+ V2) or an Authorization Key (BPI+ V1), the key's lifetime, the key's sequence number, and a list of SA-Descriptors identifying the Primary and Static SAs that the CM is authorized to access, along with their particular properties (i.e., type, cryptographic suite). The CMTS MUST securely convey the Authorization Key to the CM either by encrypting the key with the CM's public key for BPI+ V1 or by sending its key-exchange public parameters, which are used by the CM, in combination with its own, to derive the Authorization Key.

In the SA-Descriptor list, the CMTS MUST include a descriptor for the Primary SAID contained in the Authorization Request.

In the SA-Descriptor list, the CMTS MAY include descriptors of Static SAIDs that the CM is authorized to access.

The attributes of an Authorization Reply sent by the CMTS depends on the version of BPI+ used for the authorization process (see Section 7.1.3).

The CMTS MUST reply with a BPI+ V2 Authorization Reply if and only if receiving a BPI+ V2 Authorization Request.

The CMTS MUST reply with a BPI+ V1 Authorization Reply if and only if receiving a BPI+ V1 Authorization Request.

7.2.1.2.1 Common Authorization Reply Attributes

Attribute List:

Table 14 - Common Authorization Reply Attributes

Attribute	Contents	See
Key-Lifetime	Authorization Key remaining lifetime	Section 7.2.2.9
Key-Sequence-Number	Authorization Key sequence number	Section 7.2.2.10
SA-Descriptor (one or more)	Compound attribute containing SAID and other properties.	Section 7.2.2.23

7.2.1.2.2 BPI+ V1 Authorization Reply Attributes

In addition to the Common Attributes, BPI+ V1 supports the attributes described in this section.

Attribute List:

Table 15 - BPI+ V1 Authorization Reply Attributes

Attribute	Contents	See
Auth-Key	Contains the Authorization Key, RSA encrypted with public key	Section 7.2.2.7

7.2.1.2.3 BPI+ V2 Authorization Reply Attributes

In addition to the Common Attributes, BPI+ V2 supports the attributes described in this section.

Attributes List:

Table 16 - Authorization Reply Attributes (BPI+ V2)

Attribute	Contents	See
BPI-Version	BPI+ Version Number	Section 7.2.2.22
Allowed-BPI-Versions	Restrict the CM to designated BPI+ Versions (optional)	Section 7.2.2.36
CMTS-Designation	Restrict the CM to designated CMTS (optional)	Section 7.2.2.38
Key-Exchange-Share	CMTS Key Exchange Public Parameters	Section 7.2.2.35
CM-Nonce	The CM-Nonce value provided in the Authorization Request	Section 7.2.2.33
OCSP-Responses	OCSP Responses for the CMTS' certificate chain	Section 7.2.2.37
Message-Signature	Contains the message signature over the attributes	Section 7.2.2.34

The `BPI-Version` Attribute carries the version of BPI+.

The `Allowed-BPI-Versions` Attribute contains a list of BPI+ versions the CM is allowed to use for connecting on the next authorization exchange.

> NOTE: The `Allowed-BPI-Versions` Attribute does not affect the negotiated BPI+ version used for the current authorization session as described in Section 7.1.3. Specifically, this Attribute is used to allow the CM to perform authentication at a lower BPI+ version than the current one. For instance, operators can set this field to carry two values (e.g., "1" and "2") to indicate that the CM is allowed to use BPI+ V1 if support for BPI+ V2 is not detected. See Sections 7.1.4 and 12.2 for more information about downgrades attack mitigation and Trust on first Use (TOFU).

The `CMTS-Designation` Attribute(s) provide expected values for specific fields of the CMTS' certificate.

The `CM-Nonce` Attribute carries the NONCE value that ties the message to the corresponding request.

The `OCSP-Responses` Attribute carries the revocation status for the CMTS certificate chain.

7.2.1.3 Authorization Reject (Auth Reject)

Code: 6

The CMTS responds to a CM's Authorization Request with an Authorization Reject message if the CMTS rejects the CM's authorization request.

The set Attributes of an Authorization Reject sent by the CMTS depends on the version of BPI+ used in the authorization process (see Section 7.1.3) and the form of Authorization Request sent by the CM.

7.2.1.3.1 Common Authorization Reject Attributes

Attribute List:

Table 17 - Common Authorization Reject Attributes

Attribute	Contents	See
Error-Code	Error code identifying reason for rejection of Authorization Request	Section 7.2.2.15
Display-String (OPTIONAL)	Textual reason for rejection of authorization request	Section 7.2.2.6

The `Error-Code` and `Display-String` attributes describe the reason for the authorization failure.

7.2.1.3.2 BPI+ V2 Reject Attributes

In addition to the Common Attributes, BPI+ V2 supports the attributes described in this section.

Attributes List:

Table 18 - BPI+ V2 Authorization Reject Attributes

Attribute	Contents	See
BPI-Version	BPI+ Version Number	Section 7.2.2.22
CM-Nonce	The CM-Nonce value provided in the Authorization Request	Section 7.2.2.33
OCSP-Responses	OCSP Responses for the CMTS' certificate chain	Section 7.2.2.37
Message-Signature	Contains the message signature over the attributes	Section 7.2.2.34

The `BPI-Version` Attribute carries the version of BPI+ used.

The `CM-Nonce` Attribute carries the NONCE value that ties the message to the corresponding request.

The `OCSP-Responses` Attribute carries the revocation status for the CMTS certificate chain.

7.2.1.4 Key Request

Code: 7

7.2.1.4.1 Common Key Request Attributes

Attribute List:

Table 19 - Common Key Request Attributes

Attribute	Contents	See
Key-Sequence-Number	Authorization key sequence number	Section 7.2.2.10
SAID	Security Association ID	Section 7.2.2.12
HMAC-Digest	Keyed message digest	Section 7.2.2.11

The `HMAC-Digest` attribute is a keyed message digest. The CM MUST ensure that the `HMAC-Digest` attribute is the final attribute in the Key Request's Attribute list. The message digest is performed over the packet header and all the Key Request's Attribute fields, other than the `HMAC-Digest`, in the order in which they appear within the packet.

The `HMAC-Digest` authentication key is derived from the Authorization Key (see Section 11.4 for details).

7.2.1.4.2 BPI+ V1 Key Request Attributes

In addition to the Common Attributes, BPI+ V1 supports the attributes described in this section.

Attribute List:

Table 20 - BPI+ V1 Key Request Attributes

Attribute	Contents	See
CM-Identification	Contains information used to identify CM to CMTS	Section 7.2.2.5

7.2.1.4.3 BPI+ V2 Key Request Attributes

In addition to the Common Attributes, BPI+ V2 supports the attributes described in this section.

Attribute List:

Table 21 - BPI+ V2 Key Request Attributes

Attribute	Contents	See
MAC-Address	Contains information used to identify the CM to the CMTS	Section 7.2.2.3

7.2.1.5 Key Reply

Code: 8

7.2.1.5.1 Common Key Reply Attributes

Attribute List:

Table 22 - Common Key Reply Attributes

Attribute	Contents	See
Key-Sequence-Number	Authorization key sequence number	Section 7.2.2.10
SAID	Security Association ID	Section 7.2.2.12
TEK-Parameters	"Older" generation of key parameters relevant to SAID	Section 7.2.2.13
TEK-Parameters	"Newer" generation of key parameters relevant to SAID	Section 7.2.2.13
HMAC-Digest	Keyed SHA message digest	Section 7.2.2.11

The `TEK-Parameters` attribute is a compound attribute containing all the keying material corresponding to a particular generation of an SAID's TEK: the TEK, the TEK's remaining key lifetime, its key sequence number, and the CBC initialization vector. The TEK is encrypted (see Section 11.2 for details).

The `HMAC-Digest` attribute is a keyed message digest.

The CMTS MUST ensure that the `HMAC-Digest` attribute is the final attribute in the Key Reply's Attribute list.

The message digest is performed over the BPKM message header (starting with the BPKM `Code` field) and all of the Key Reply's Attribute fields, other than the `HMAC-Digest`, in the order in which they appear within the packet.

The `HMAC-Digest` authentication key is derived from the Authorization Key (see Section 11.4 for details).

7.2.1.6 Key Reject

Code: 9

Receipt of a Key Reject indicates that the recipient CM is no longer authorized to use a particular SAID.

7.2.1.6.1 Common Key Reject Attributes

Attribute List:

Table 23 - Common Key Reject Attributes

Attribute	Contents	See
Key-Sequence-Number	Authorization key sequence number	Section 7.2.2.10
SAID	Security Association ID	Section 7.2.2.12
Error-Code	Error code identifying reason for rejection of Key Request	Section 7.2.2.15
Display-String (OPTIONAL)	Display string containing reason for Key Reject	Section 7.2.2.6
HMAC-Digest	Keyed SHA message digest	Section 7.2.2.11

The `HMAC-Digest` attribute is a keyed message digest.

The CMTS MUST ensure that the `HMAC-Digest` attribute is the final attribute in the Key Reject's Attribute list.

The message digest is performed over the BPKM message header (starting with the BPKM `Code` field) and all of the Key Reject's Attribute fields, other than the `HMAC-Digest`, in the order in which they appear within the packet. The `HMAC-Digest` authentication key is derived from the Authorization Key (see Section 11.4 for details).

7.2.1.7 Authorization Invalid

Code: 10

The Authorization Invalid message instructs the receiving CM to reauthorize with its CMTS.

7.2.1.7.1 Common Authorization Invalid Attributes

Attribute List:

Table 24 - Common Authorization Invalid Attributes

Attribute	Contents	See
Error-Code	Error code identifying reason for Authorization Invalid	Section 7.2.2.10
Display-String (OPTIONAL)	Textual description of failure condition	Section 7.2.2.6

7.2.1.8 TEK Invalid

Code: 11

The CMTS sends a TEK Invalid message to a client CM if the CMTS determines that the CM encrypted an upstream PDU with an invalid TEK: i.e., an SAID's TEK key sequence number, contained within the received packet's Baseline Privacy EH element, is out of the CMTS's range of known, valid sequence numbers for that SAID.

7.2.1.8.1 Common TEK Invalid Attributes

Attribute List:

Table 25 - Common TEK Invalid Attributes

Attribute	Contents	See
Key-Sequence-Number	Authorization key sequence number	Section 7.2.2.10
SAID	Security Association ID	Section 7.2.2.12
Error-Code	Error code identifying reason for TEK Invalid message	Section 7.2.2.15
Display-String (OPTIONAL)	Textual vendor-defined information	Section 7.2.2.6
HMAC-Digest	Keyed SHA message digest	Section 7.2.2.11

The `HMAC-Digest` attribute is a keyed message digest.

The CMTS MUST ensure that the HMAC-Digest attribute is the final attribute in the TEK Invalid's Attribute list.

The message digest is performed over the BPKM message header (starting with the BPKM Code field) and all of the TEK Invalid's attribute fields, other than the HMAC-Digest, in the order in which they appear within the packet. The HMAC-Digest authentication key is derived from the Authorization Key (see Section 11.4 for details).

7.2.1.9 Authentication Information (Auth Info)

Code: 12

The Authentication Info message contains a single CA-Certificate Attribute field, which contains an Intermediate (or Subordinate) CA Certificate.

The CM MUST use the Device Certificate issued by the CA identified by this certificate to authenticate itself to the network.

Authentication Information messages are usually informative: although the CM is required to transmit Auth Info messages as indicated by the authorization state machine (see Section 7.1.6), the CMTS under some circumstances may ignore them.

7.2.1.9.1 Common Authentication Information Attributes

Attribute:

Table 26 - Common Authentication Information Attributes

Attribute	Contents	See
CA-Certificate	Certificate of the legacy Manufacturer CA or Device CA	Section 7.2.2.17 and Section 7.2.2.31

7.2.1.10 SA Map Request (MAP Request) [2]

Code: 13

A CM sends SA Map Requests to its CMTS to request the mapping of a particular downstream traffic flow to an SA. Annex C describes the SA Mapping state model.

7.2.1.10.1 Common SA Map Request Attributes

Attribute List:

Table 27 - Common SA Map Request Attributes

Attribute	Contents	See
SA-Query	Contains addressing information identifying the downstream traffic flow for which the CM is requesting an SA mapping	Section 7.2.2.25

7.2.1.10.2 BPI+ V1 SA Map Request Attributes

In addition to the Common Attributes, BPI+ V1 supports the attributes described in this section.

Table 28 - BPI+ V1 SA Map Request Attributes

Attribute	Contents	See
CM-Identification	Contains information used to identify the CM to the CMTS	Section 7.2.2.5

7.2.1.10.3 BPI+ V2 SA Map Request Attributes

In addition to the Common Attributes, BPI+ V2 supports the attributes described in this section.

[2] The SA mapping mechanism was used in earlier versions of DOCSIS technology to support Dynamic SAs (see Annex C).

Table 29 - BPI+ V2 SA Map Request Attributes

Attribute	Contents	See
MAC-Address	Contains information used to identify the CM to the CMTS	Section 7.2.2.3

7.2.1.11 SA Map Reply (Map Reply)

Code: 14

A CMTS sends an SA Map Reply as a positive response to a client CM's SA Map Request. The SA Map Reply informs the CM of a mapping between a queried address and an SA. Annex C describes the SA Mapping state model.

Attribute List:

Table 30 - Common SA Map Reply Attributes

Attribute	Contents	See
SA-Query	Contains addressing information identifying the downstream traffic flow for which the CM is requested an SA mapping	Section 7.2.2.25
SA-Descriptor	Compound attribute containing the mapped SA's SAID and other properties	Section 7.2.2.23

7.2.1.12 SA Map Reject (Map Reject)

Code: 15

A CMTS sends an SA Map Reject as a negative response to a client CM's SA Map Request. The SA Map Reject informs the CM that either

- downstream traffic flow identified in the `SA-Query` attribute is not being encrypted or
- the requesting CM is not authorized to receive that traffic.

The content of an error code attribute distinguishes between these cases. Annex C describes the SA Mapping state model.

7.2.1.12.1 Common SA Map Reject ((Map Reject) Attributes

Attribute List:

Table 31 - Common SA MAP Reject Attributes

Attribute	Contents	See
SA-Query	Contains addressing information identifying the downstream traffic flow for which the CM requested an SA mapping	Section 7.2.2.25
Error-Code	Error code identifying reason for rejection of the SA Map Request	Section 7.2.2.15
Display-String (OPTIONAL)	Textual reason for the SA Map Reject	Section 7.2.2.6

7.2.1.13 Authentication Status Info (Auth Status Info)

Code: 16

In BPI+ V2, the CM responds to a CMTS's Authorization Reply with an Authentication Status Info message.

7.2.1.13.1 BPI+ V2 Auth Status Info Attributes

BPI+ V2 supports the attributes described in this section.

Attributes List:

Table 32 - BPI+ V2 Authorization Status Info Attributes

Attribute	Contents	See
CM-Status-Code	Error code identifying reason for rejection of Authorization Reply	Section 0
Display-String (OPTIONAL)	Textual reason for rejection of the Authorization Reply	Section 7.2.2.6

Attribute	Contents	See
Detected-Errors (OPTIONAL)	Total number of authentication issues detected by the CM	Section 7.2.2.40
BPI-Version	BPI+ Version Number	Section 7.2.2.22
CM-Nonce	The CM-Nonce value provided in the Authorization Request	Section 7.2.2.33
Message-Signature	Contains the message signature over the attributes	Section 7.2.2.34

The CM-Status-Code and Display-String attributes describe the reason for the status of the authentication and validation over the Auth Reply message.

The Message-Signature can be calculated using the procedures described in Section 7.2.2.34.

7.2.2 BPKM Attributes

A summary of the format of the Attribute field is shown below.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|    Type       |            Length             |  Value ...
+---------------+---------------+---------------+---------------+
```

Type

The Type field is one octet. Values of the BPKM Type field are specified below. Values between 0 and 127 inclusive are defined within this specification; values between 128 and 255 are vendor-assigned Attribute Types.

The CMTS MUST ignore attributes with an unknown value of Type field. The CMTS MAY log receipt of attributes with unknown values of the Type field.

The CM MUST ignore attributes with an unknown value of Type field. The CM MAY log receipt of attributes with unknown values of the Type field.

Some BPKM attributes are also used for the Secure Software Download code file (see Section 14).

Table 33 - BPKM Attribute Types

Type	BPKM Attribute
0	Reserved
1	Serial-Number
2	Manufacturer-ID
3	MAC-Address
4	RSA-Public-Key (also used for secure software download, see Section 14)
5	CM-Identification
6	Display-String
7	Auth-Key
8	TEK
9	Key-Lifetime
10	Key-Sequence-Number
11	HMAC-Digest
12	SAID
13	TEK-Parameters
14	Reserved
15	CBC-IV
16	Error-Code
17	CA-Certificate (also used for secure software download, see Section 14)
18	CM-Certificate

Type	BPKM Attribute
19	Security-Capabilities
20	Cryptographic-Suite
21	Cryptographic-Suite-List
22	BPI-Version
23	SA-Descriptor
24	SA-Type
25	SA-Query
26	SA-Query-Type
27	IPv4-Address
28	Download-Parameters (used for secure software download; see Section 14)
29-50	Reserved
51	CVC-Root-CA-Certificate (used for secure software download; see Section 14)
52	CVC-CA-Certificate (used for secure software download; see Section 14)
53	Device-CA-Certificate (used for secure software download; see Section 14)
54	Root-CA-Certificate (used for secure software download; see Section 14)
55-60	Reserved
61	CM-Nonce
62	Message-Signature
63	Key-Exchange-Share
64	Allowed-BPI-Versions
65	OCSP-Responses
66	CMTS-Designation
67	CM-Status-Code
68	Detected-Errors
69-126	Reserved
127	Vendor-Defined
128-255	Vendor-assigned attribute types

Length

The Length field is two octets and indicates the length of this attribute's Value field, in octets. The value of the Length field does not include the length of the Type and Length fields.

The CMTS MUST set the value of the Length field to be in the range [0, 1487].

The CM MUST set the value of the Length field to be in the range [0, 1487].

The CMTS SHOULD silently discard packets containing attributes with invalid values of the Length field.

The CM SHOULD silently discard packets containing attributes with invalid values of the Length field.

Value

The Value field is zero or more octets and contains information specific to the particular attribute. The format and length of the Value field is determined by the contents of the Type and Length fields. All multi-octet integer quantities are in network order, i.e., the octet containing the most-significant bits is the first transmitted on the wire.

The format of the Value field is one of five data types, as shown in Table 34.

Table 34 - Attribute Value Data Types

Type	Meaning
string	0-1487 octets

Type	Meaning
uint8	8-bit unsigned integer
uint16	16-bit unsigned integer
uint32	32-bit unsigned integer
compound	collection of attributes

7.2.2.1 Serial-Number

This attribute contains the serial identifier assigned by the manufacturer to a CM.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|    Type = 1   |         0 ≤ Length ≤ 255      | string ...
+---------------+---------------+---------------+---------------+
```

`string`

The `string` field is zero or more octets and contains a serial identifier assigned by the manufacturer.

The CM MUST encode the serial identifier in the [ISO 8859-1] character-set encoding.

The CM MUST use only the following characters:

- uppercase letters, A to Z (0x41-0x5A);
- lowercase letters, a to z (0x61-0x7A);
- digits, 0 to 9 (0x30-0x39); and
- dash, - (0xD2).

7.2.2.2 Manufacturer-ID

This attribute identifies the manufacturer. The identifier is 3 octets long and contains the 24-bit Organizationally Unique Identifier (OUI) assigned to applying organizations by the IEEE.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|    Type = 2   |            Length = 3         | string ...
+---------------+---------------+---------------+---------------+
```

`string`

The `string` field is 3 octets in length and contains an IEEE OUI.

7.2.2.3 MAC-Address

This attribute identifies the MAC address assigned to the CM.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|    Type = 3   |            Length = 6         | string ...
+---------------+---------------+---------------+---------------+
```

`string`

The `string` field contains a 6-octet MAC address.

7.2.2.4 RSA-Public-Key

This attribute is a `string` attribute containing a DER-encoded RSAPublicKey ASN.1 type, as defined in [X.509]:

```
RSAPublicKey ::= SEQUENCE {
                   modulus INTEGER,
                   publicExponent INTEGER }
```

The CMTS MUST support a publicExponent of F_4. The CM MUST support a publicExponent of F_4.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|    Type = 4   |            Length             | string ...
+---------------+---------------+---------------+---------------+
```

`Length`

The length of the DER-encoded RSAPublicKey.

`string`

DER-encoded RSAPublicKey. For example, using F4 as the public exponent, the size in octets of public keys is 106, 140, or 270 for RSA keys of 768-bit, 1024-bit, or 2048-bit public modulus, respectively.

7.2.2.5 CM-Identification

This is a compound attribute consisting of a collection of sub-attributes. These sub-attributes contain information that can be used to uniquely identify a CM.

The CM MUST provide the following sub-attributes:

- Serial-Number,
- Manufacturer-ID,
- MAC-Address, and
- RSA-Public-Key (BPI+ V1 only).

`CM-Identification` MAY also contain optional Vendor-Defined Attributes.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|    Type = 5   |          Length ≥ 126         | compound
+---------------+---------------+---------------+---------------+
```

7.2.2.6 Display-String

This attribute contains a textual message. It is typically used to explain a failure response and might be logged by the receiver for later retrieval by an SNMP manager.

The CMTS MUST NOT use display strings longer than 128 octets.

The CM MUST NOT use display strings longer than 128 octets.

This specification does not define the character set or the language to be used in the `Display-String` attribute.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|    Type = 6   |       0 ≤ Length ≤ 128        | string...
+---------------+---------------+---------------+---------------+
```

`string`

A string of octets.

7.2.2.7 Auth-Key

The Authorization Key is a twenty (20)-octet value from which other keys are derived.

This attribute is used in BPI+ V1 only and it contains an octet value of either ninety-six (96), one-hundred twenty-eight (128), or two-hundred fifty-six (256) that is the Authorization Key encrypted with the CM's 768-bit, 1024-bit, or 2048-bit RSA public key. Details of the RSA encryption procedure are given in Section 11.5.1.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|    Type = 7   |    Length = 96,128 or 256     | string ...
+---------------+---------------+---------------+---------------+
```

`string`

Encrypted Authorization Key.

7.2.2.8 TEK

This attribute contains a TEK encrypted with a KEK derived from the Authorization Key. TEKs are encrypted using the encrypt-decrypt-encrypt (EDE) mode of two-key 3DES (see Section 11.2 for details).

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|    Type = 8   |             Length            | string ...
+---------------+---------------+---------------+---------------+
```

`Length`

Eight (8) (DES TEK) or sixteen (16) (AES TEK).

`string`

Encrypted Traffic Encryption Key.

7.2.2.9 Key-Lifetime

This attribute contains the lifetime, in seconds, of an Authorization Key or TEK. It is a thirty-two (32)-bit unsigned quantity representing the remaining number of seconds for which the associated key is valid.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|    Type = 9   |           Length = 4          |   uint32 ...  |
+---------------+---------------+---------------+---------------+
|              ... uint32                       |
+---------------+---------------+---------------+
```

`uint32`

Remaining key lifetime, in seconds.

A key lifetime of zero indicates that the corresponding key is not valid.

7.2.2.10 Key-Sequence-Number

This attribute contains a 4-bit sequence number for a TEK or Authorization Key. The four (4)-bit quantity is stored in an octet.

The CMTS MUST set the high-order four (4) bits to zero (0).

The CM MUST set the high-order four (4) bits to zero (0).

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|    Type = 10  |            Length = 1         |     uint8     |
+---------------+---------------+---------------+---------------+
```

`uint8`

4-bit sequence number.

7.2.2.11 HMAC-Digest

This attribute contains a keyed hash used for message authentication. The HMAC algorithm is defined in [RFC 2104]; the hash algorithm is SHA-1 [FIPS 180-4].

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|    Type = 11  |           Length = 20         | string ...
+---------------+---------------+---------------+---------------+
```

`string`

A 160-bit (20-octet) keyed SHA-1 hash.

7.2.2.12 SAID

This attribute contains a fourteen (14)-bit SAID.

The CMTS MUST set the two high-order bits to zero.

The CM MUST set the two high-order bits to zero.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|    Type = 12  |            Length = 2         |  uint16 ...   |
+---------------+---------------+---------------+---------------+
|    ...uint16  |
+---------------
```

`uint16`

An SAID.

7.2.2.13 TEK-Parameters

This is a compound attribute consisting of a collection of sub-attributes. These sub-attributes represent all the security parameters relevant to a particular generation of an SAID's TEK.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|    Type = 13  |           Length        | compound ...
+---------------+---------------+---------------+---------------+
```

`Length`

Thirty-three (33) (DES) or forty-nine (49) (AES).

`compound`

This field contains the sub-attributes listed in Table 35.

Table 35 - TEK-Parameters Sub-Attributes

Attribute	Contents	See
TEK	TEK, encrypted (two-key 3DES-EDE mode) with the KEK	Section 7.2.2.8
Key-Lifetime	TEK Remaining Lifetime	Section 7.2.2.9
Key-Sequence-Number	TEK Sequence Number	Section 7.2.2.10
CBC-IV	Cipher block chaining (CBC) initialization vector	Section 7.2.2.14

7.2.2.14 CBC-IV

This attribute contains a CBC initialization vector.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|   Type = 15   |            Length             | string ...
+---------------+---------------+---------------+---------------+
```

`Length`

Eight (8) (DES) or sixteen (16) (AES).

`string`

Initialization vector.

7.2.2.15 Error-Code

This attribute contains a 1-octet error code that provides further information about an Authorization Reject, Key Reject, Authorization Invalid, Authentication Status Info, SA-MAP Reject,[3] or TEK Invalid message.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|   Type = 16   |          Length = 1           |     uint8     |
+---------------+---------------+---------------+---------------+
```

`uint8`

1-octet error code.

The CMTS MUST include the Error-Code Attribute in all Authorization Reject, Authorization Invalid, Key Reject, TEK Invalid, and SA-MAP Reject messages.

Table 36 lists code values for use with this Attribute.

The CMTS MUST employ the non-zero error codes listed in the table for SA-MAP Reject messages.

The CMTS MUST employ error codes listed in Table 36 - Error-Code Attribute Code Values (including zero) for the other message types.

The CM MUST ignore error code values other than those defined in Table 36 - Error-Code Attribute Code Values.

Table 36 - Error-Code Attribute Code Values

Error Code	Messages	Description
0	All	No Information
1	Auth Reject, Auth Invalid	Unauthorized CM
2	Auth Reject, Key Reject	Unauthorized SAID
3	Auth Invalid	Unsolicited

[3] The SA mapping mechanism was used in earlier versions of DOCSIS technology to support Dynamic SAs (see Annex C).

Error Code	Messages	Description
4	Auth Invalid, TEK Invalid	Invalid Key Sequence Number
5	Auth Invalid	Message (Key Request) authentication failure
6	Auth Reject	Permanent Authorization Failure
7	Map Reject	Not authorized for requested downstream traffic flow
8	Map Reject	Downstream traffic flow not mapped to SAID
9	Auth Reject	Time of day not acquired
10	Auth Reject	EAE Disabled
11	Auth Reject	BPI+ Version not supported

Error code 6, Permanent Authorization Failure, is used to indicate a number of different error conditions affecting the BPKM authorization exchange. These include the following:

- an unknown manufacturer, i.e., the CMTS does not have the CA certificate belonging to the issuer of a CM Device Certificate;
- CM Device Certificate has an invalid signature;
- ASN.1 parsing failure during verification of CM Device Certificate;
- CM Device Certificate is revoked (see Section 13.4);
- inconsistencies between certificate data and data in accompanying BPKM attributes; and
- the CM and CMTS have incompatible security capabilities.

The CMTS MUST send an Authorization Reject message containing error code 6 (Permanent Authorization Failure) in response to an Authorization Request message when any of the following occurs:

- the CMTS fails to validate the CM Device Certificates (see Section 13.3.2, Certificate Validation) or
- the CM and the CMTS have incompatible security capabilities (see Section 7.2.1.1, Authorization Request (Auth Request)).

Entries in the CMTS MIB (see [DOCSIS CM-OSSIv4.0]) control the actions that a CMTS takes in the event that any of the above error conditions occur.

The CMTS MAY report details about the cause of a Permanent Authorization Failure to the CM in an optional Display-String Attribute that may accompany the Error-Code Attribute in Authorization Reject messages.

The CMTS SHOULD provide the capability to control administratively whether additional detail is sent to the CM.

The CMTS MAY log these Authorization failures or otherwise make them known to the operator.

7.2.2.16 Vendor-Defined

The Vendor-Defined attribute is a compound attribute whose first sub-attribute is the Manufacturer-ID. Subsequent attribute(s) are defined by the manufacturer, with Type values assigned by the vendor identified by the Manufacturer-ID.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|    Type = 127 |           Length  ≥ 6         | compound ...
+---------------+---------------+---------------+---------------+
```

`compound`

The CMTS MUST insert the Manufacturer-ID as the first sub-attribute. The CM MUST insert the Manufacturer-ID as the first sub-attribute.

Subsequent attributes can include both Types defined within this specification and vendor-defined Types, defined by the vendor identified in the preceding Manufacturer-ID sub-attribute.

7.2.2.17 CA-Certificate

This attribute contains a DER-encoded Device CA Certificate.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|    Type = 17  |             Length            | string ...
+---------------+---------------+---------------+---------------+
```

Length

Variable.

string

DER-encoded Device CA certificate

7.2.2.18 CM-Certificate

This attribute contains the DER-encoded CM Device Certificate.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|    Type = 18  |             Length            | string ...
+---------------+---------------+---------------+---------------+
```

Length

Variable.

string

DER-encoded CM Device Certificate.

7.2.2.19 Security-Capabilities

This is a compound attribute whose sub-attributes identify the version of DOCSIS Security and cryptographic suite(s) supported by a CM.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|    Type = 19  |            Length ≥ 9         | compound ...
+---------------+---------------+---------------+---------------+
```

compound

This field contains the sub-attributes listed in Table 37.

Table 37 - Security-Capabilities Sub-Attributes

Attribute	Contents	See
Cryptographic-Suite-List	List of supported cryptographic suites	7.2.2.21
BPI-Version	Version of BPI+ supported	7.2.2.22

7.2.2.20 Cryptographic-Suite

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +---------------+---------------+---------------+---------------+
 |    Type = 20  |           Length = 2          |  uint16 ...   |
 +---------------+---------------+---------------+---------------+
 |    ...uint16  |
 +---------------+
```

uint16

A 16-bit integer identifying a pairing of a data encryption algorithm (encoded in the most significant octet) taken from Table 38 and a data authentication algorithm (encoded in the least significant octet) taken from Table 39.

The CMTS MUST use a value that appears in Table 40 - Cryptographic-Suite Attribute Values.

The CM MUST use a value that appears in Table 40 - Cryptographic-Suite Attribute Values.

Table 38 - Data Encryption Algorithm Identifiers

Value	Description
0	Reserved
1	CBC-Mode, 56-bit DES
2	CBC-Mode, 40-bit DES
3	CBC-Mode, 128-bit block, 128-bit key AES
4	CBC-Mode, 128-bit block, 256-bit key AES
5-255	Reserved

Table 39 - Data Authentication Algorithm Identifiers

Value	Description
0	No data authentication
1-255	Reserved

Table 40 - Cryptographic-Suite Attribute Values

Value	Description
256 (0x0100)	CBC-Mode 56-bit DES, no data authentication
512 (0x0200)	CBC-Mode 40-bit DES, no data authentication
768 (0x0300)	CBC-Mode 128-bit AES, no data authentication
1024 (0x0400)	CBC-Mode 256-bit AES, no data authentication
All remaining values	Reserved

7.2.2.21 Cryptographic-Suite-List

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +---------------+---------------+---------------+---------------+
 |    Type = 21  |              Length           | string ...
 +---------------+---------------+---------------+---------------+
```

Length

2 × n, where n is the number of cryptographic suites in the list.

string

A list of byte pairs identifying a collection of cryptographic suites. Each byte pair represents a supported cryptographic suite, with an encoding identical to the value field of the Cryptographic-Suite Attribute (Section

7.2.2.20). The CMTS MUST NOT interpret the relative ordering of byte pairs in the list as a CM's preferences amongst the cryptographic suites it supports.

7.2.2.22 BPI-Version

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|    Type = 22  |          Length = 1           |     uint8     |
+---------------+---------------+---------------+---------------+
```

`uint8`

A one (1)-octet code identifying a version of Baseline Privacy security.

Table 41 - BPI-Version Attribute Values

Value	Description
0	Reserved
1	BPI+ V1
2	BPI+ V2
3-255	Reserved

7.2.2.23 SA-Descriptor

This is a compound attribute whose sub-attributes describe the properties of an SA. These properties are the SAID, the SA type, and the cryptographic suite employed by the SA.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|    Type = 23  |          Length = 14          | compound ...
+---------------+---------------+---------------+---------------+
```

`compound`

This field contains the sub-attributes shown in Table 42.

Table 42 - SA-Descriptor Sub-Attributes

Attribute	Contents	See
SAID	Security Association ID	7.2.2.12
SA-Type	Type of SA	7.2.2.24
Cryptographic-Suite	Pairing of data encryption and data authentication algorithms employed within the SA	7.2.2.20

7.2.2.24 SA-Type

Identifies the type of SA. This specification defines three types of SA: Primary, Static, and Dynamic.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|    Type = 24  |           Length = 1          |     uint8     |
+---------------+---------------+---------------+---------------+
```

`uint8`

A 1-octet code identifying the value of SA-type as defined in Table 43.

Table 43 - SA-Type Attribute Values

Value	Description
0	Primary
1	Static
2	Dynamic
3-255	Reserved

7.2.2.25 SA-Query [4]

This compound attribute is used in SA Map Request messages to pass mapping query arguments. Query arguments include the query type and any addressing attributes particular to that query type. The addressing attributes identify a particular downstream traffic flow for which an SA mapping is being requested.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|    Type = 25  |           Length = 11         | compound ...
+---------------+---------------+---------------+---------------+
```

`compound`

This field contains the sub-attributes in Table 44.

Table 44 - SA-Query Sub-Attributes

Attribute	Contents	See
SA-Query-Type	Type of Query	7.2.2.26
IPv4-Address	Required if SA-Query-Type = IP-Multicast; contains an IPv4 group address whose SA mapping is being requested.	7.2.2.27

7.2.2.26 SA-Query-Type

Identifies the type of query. This specification defines two types of SA Queries: IP Multicast and Vendor Specific.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|    Type = 26  |           Length = 1          |     uint8     |
+---------------+---------------+---------------+---------------+
```

`uint8`

A 1-octet code identifying the value of SA-Query-Type as defined in Table 45.

Table 45 - SA-Query-Type Attribute Values

Value	Description
0	Reserved
1	IP Multicast
2-127	Reserved
128-255	Vendor Specific

[4] The SA mapping mechanism was used in earlier versions of DOCSIS technology to support Dynamic SAs (see Annex C).

7.2.2.27 IPv4-Address

This attribute identifies an IPv4 address that is used to identify an encrypted IP traffic flow. It is used, for example, to specify an IPv4 multicast group address.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|    Type = 27  |            Length = 4         | uint32 ...
+---------------+---------------+---------------+---------------+
```

`uint32`

Contains a thirty-two (32)-bit unsigned integer (in network order) representing an IPv4 address.

7.2.2.28 Download-Parameters

This attribute is used in the CM Code File defined in Section 14. This is a compound attribute consisting of an ordered collection of sub-attributes.

The sub-attributes include zero or more of the following attribute(s) in this order.

New PKI Attributes

- Zero or one instance of Device-CA-Certificate (see Section 7.2.2.31)
- Zero or one instance of Root-CA-Certificate (see Section 7.2.2.32)

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|    Type = 28  |            Length ≥ 0         | compound ...
+---------------+---------------+---------------+---------------+
```

7.2.2.29 CVC-Root-CA-Certificate (deprecated)

This attribute contains a DER-encoded CVC Root CA certificate from the legacy PKI.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|    Type = 51  |             Length            | string ...
+---------------+---------------+---------------+---------------+
```

`Length`

Variable.

`string`

DER-encoded CVC Root CA Certificate.

7.2.2.30 CVC-CA-Certificate (deprecated)

This attribute contains a DER-encoded CVC CA certificate from the legacy PKI.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|    Type = 52  |             Length            | string ...
+---------------+---------------+---------------+---------------+
```

`Length`

Variable.

`string`

DER-encoded CVC CA Certificate.

7.2.2.31 Device-CA-Certificate

This attribute contains a DER-encoded Device CA Certificate from the new PKI.

```
       0                   1                   2                   3
       0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
      +---------------+---------------+---------------+---------------+
      |    Type = 53  |             Length            | string ...
      +---------------+---------------+---------------+---------------+
```

`Length`

Variable.

`string`

DER-encoded Device CA Certificate from the new PKI.

7.2.2.32 Root-CA-Certificate

This attribute contains a DER-encoded Root CA Certificate from the new PKI.

```
       0                   1                   2                   3
       0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
      +---------------+---------------+---------------+---------------+
      |    Type = 54  |             Length            | string ...
      +---------------+---------------+---------------+---------------+
```

`Length`

Variable.

`string`

DER-encoded Root CA Certificate from the new PKI.

7.2.2.33 CM-Nonce

```
       0                   1                   2                   3
       0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
      +---------------+---------------+---------------+---------------+
      |    Type = 61  |             Length            | string  ...
      +---------------+---------------+---------------+---------------+
```

`Length`

Fixed (8 octets)

`string`

A string of 8-octets value used to tie Auth Req and Auth Reply. The value should be random enough to limit the possibility of collisions (same NONCE used more than once) in order to prevent Replay Attacks.

7.2.2.34 Message-Signature

This attribute contains a DER-encoded CMS signature over the header and attributes in a BPKM message. The signature is created using the private key corresponding to the signer's certificate (e.g., the CM's or the CMTS's private key) and a SignedData structure as defined in the Cryptographic Message Syntax (CMS) [RFC 5652].

```
       0                   1                   2                   3
       0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
      +---------------+---------------+---------------+---------------+
      |    Type = 62  |           Length              |   string ...
      +---------------+---------------+---------------+---------------+
```

`Length`

Variable

String (variable)

The SignedData is calculated over the Attributes and message headers of the BPKM message it is embedded into in the same order they appear in the message (except the Message-Signature Attribute itself). The Message Signature attribute is the last attribute in the message.

> NOTE: Since the signature is calculated over the header and Attributes of a BPKM, the length of the message is included in the signature calculation. The length field carries the overall length of the message including the Message-Signature attribute.

The SignedData structure has to include:

- the signerInfo structure related to the signer.
- the certificates field that at minimum includes the signer's certificate and can also include the issuing CA certificate.

The CMS does not include the signed content in the SignedData; specifically, the EncapsulatedContentInfo should omit the eContent optional field.

7.2.2.35 Key-Exchange-Share

This attribute contains one key share to be used to derive the BPI+ V2 authorization key. The TLV comprises two different fields that are used to convey the ECDHE field (uint16) and the specific value for the key share (variable).

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +---------------+---------------+---------------+---------------+
 |    Type = 63  |           Length              |    fieldId   ...
 +---------------+---------------+---------------+---------------+
...              |          keyShare                             ...
 +---------------+---------------+---------------+---------------+
```

Length

Variable

fieldId (uint16)

The fieldId provides the indication of the group used for the keyShare generation (i.e., elliptic curve groups for ECDHE).

When replying to Key-Exchange-Share (e.g., when the CMTS sends the BPI+ V2 Auth Reply to the CM), the fieldId MUST be the same as the one used to calculate the first half of the key exchange parameters.

The supported values for Elliptic-Curve Groups (ECDHE):

- secp256r1 (0x0017)
- secp384r1 (0x0018)
- secp521r1 (0x0019)
- x25519 (0x001D)
- x448 (0x001E)

keyShare (variable)

The keyShare field carries the structure for the ECDHE parameters and the parameters are encoded as described in Section 4.2.8.2 of [RFC 8446]. All ECDH operations (including shared secret calculation) are performed as described in Section 7.4.2 of [RFC 8446].

7.2.2.36 *Allowed-BPI-Versions*

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+--------------+
|    Type = 64  |             Length            | string  ...
+---------------+---------------+---------------+--------------+
```

`Length`

One octet per allowed BPI+ version.

`string`

A list of `uint8`s identifying the BPI+ Versions supported. See Section 7.2.2.22 for a definition of the BPI+ Version codes.

7.2.2.37 *OCSP-Responses*

This attribute contains the OCSP responses of the CMTS certificate and, optionally, its issuing CA certificate. The CM uses the `OCSP-Responses` attribute to check the revocation status of the CMTS certificate and its issuing CA certificate(s) contained in the `Message-Signature` attribute.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|    Type = 65  |            Length             |    string    ...
+---------------+---------------+---------------+---------------+
```

`Length`

Variable

`string`

This field carries the OCSP responses authoritative for the CMTS Device Certificate and its certificate chain.

The string value contains the concatenation of the DER encoded `OCSPResponse` messages for the CMTS Certificate chain.

The field carries the concatenation of the OCSP responses starting from the End-Entity Certificate. The OCSP message format and message processing rules are defined in [RFC 6960].

7.2.2.38 *CMTS-Designation*

This attribute is used to provide rules to bind a CM to a CMTS. Multiple rules can be added to provide the level of granularity desired.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|    Type = 66  |             Length            |   dataType    |
+---------------+---------------+---------------+---------------+
|    Value     ...
+               +
```

Variable.

`dataType (uint8)`

This field indicates the type of data to be checked. The allowed values are listed in Table 46.

`Value (string)`

This field provides the value to be used when checking the authorization parameters at the next authorization session. For example, when the operator wants to restrict the connectivity to only a specific organization (e.g., "Cable Co.") it configures the CMTS to use a CMTS-Designation attribute with the inner Type set to 0x01 (1) and the Value set to the expected value for the organization (O) field of Subject of the CMTS Device Certificate (e.g., "Cable Co.").

Table 46 - CMTS Binding Attributes Names

DataType	Attributes	Validation Steps
0	Certificate-Fingerprint	The value of the SHA-256 digest calculated over the DER representation of the CMTS certificate.
1	Common-Name	A value of a CMTS certificate's common name
2	Organizational-Unit	A value of the CMTS certificate's organization unit
3	Organization-Name	A value of the CMTS certificate's organization name
4	Serial-Number	The value of the CMTS Device CA certificate's serial number
5	Issuing-CA-Fingerprint	The value of the SHA-256 digest calculated over the DER representation of the CMTS Device CA certificate
6	Issuing-CA-Common-Name	A value of the CMTS Device CA certificate's common name
7	Issuing-CA-Organizational-Unit	A value of the CMTS Device CA certificate's organization unit
8	Issuing-CA-Organization-Name	A value of the CMTS Device CA certificate's organization name
9	Issuing-CA-Serial-Number	The value of the CMTS Device CA certificate's serial number
10-255	Reserved	n/a

The following requirements apply to Table 46:

The CM MUST verify that the CMTS `Certificate-Fingerprint` (SHA-256 calculated over the DER representation of the certificate) matches the value carried in the `value` field.

The CM MUST verify that at least one of the CMTS certificate `Common-Name` matches the value carried in the `value` field.

The CM MUST verify that at least one of the CMTS certificate’s `Organizational-Unit` entries matches the value carried in the `value` field.

The CM MUST verify that at least one of the CMTS certificate’s `Organization-Name` matches the value carried in the `value` field.

The CM MUST verify that the CMTS certificate’s `Serial-Number` matches the value carried in the `value` field.

The CM MUST verify that the CMTS certificate's `Issuing-CA-Fingerprint` (SHA-256 calculated over the DER representation of the certificate) matches the value carried in the `value` field.

The CM MUST verify that at least one of the CMTS certificate's `Issuing-CA-Common-Name` matches the value carried in the `value` field.

The CM MUST verify that at least one of the CMTS certificate's `Issuing-CA-Organizational-Unit` matches the value carried in the `value` field.

The CM MUST verify that at least one of the CMTS certificate's `Issuing-CA-Organization-Name` matches the value carried in the `value` field.

The CM MUST verify that the CMTS certificate's `Issuing-CA-Serial-Number` matches the value carried in the `value` field.

7.2.2.39 CM-Status-Code

This attribute contains a 1-octet error code that provides further information about the status of the authentication performed by the CM on the Auth Reply from the CMTS.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|    Type = 67  |            Length = 1         |    uint8      |
+---------------+---------------+---------------+---------------+
```

`uint8`

1-octet error code.

The CM MUST include the CM-Status-Code Attribute in all Authentication Status Info messages.

Table 47 lists code values for use with this Attribute.

Table 47 - CM-Status-Code Attribute Code Values

Status Code	Description
0	No error
1	Generic error
2	Auth Reply not received
3	Missing Auth Reply required attribute
4	BPI-Version mismatch
5	NONCE mismatch
6-10	Reserved
11	Signature Format Error
12	Signature Algorithm not supported
13	Public Key Algorithm not supported
14	Incomplete Certificate Chain
15	Certificate Not Trusted
16	Missing EE certificate revocation information
17	Missing CA certificate revocation information
18	EE certificate Expired
19	CA certificate Expired
20	CMTS-designation fingerprint (SHA-256) mismatch
21	CMTS-designation Common-Name mismatch
22	CMTS-designation Organizational-Unit mismatch
23	CMTS-designation Organization-Name mismatch
24	CMTS-designation Serial-Number mismatch
25	CMTS-designation Issuing-CA-fingerprint (SHA-256) mismatch
26	CMTS-designation Issuing-CA-Common-Name mismatch
27	CMTS-designation Issuing-CA-Organizational-Unit mismatch
28	CMTS-designation Issuing-CA-Organization mismatch
29	CMTS-designation Issuing-CA-Serial-umber mismatch
30	Missing Key-Derivation required parameters
31	Key-Derivation parameters field mismatch
32	Key-Derivation error
33-255	Reserved

7.2.2.40 Detected-Errors

This attribute contains the number of errors detected by the CM in addition to the issue associated with the reported `CM-Status-Code`.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|    Type = 68  |            Length = 1         |     uint8     |
+---------------+---------------+---------------+---------------+
```

`uint8`

Indicates the number of detected errors on the CM.

8 EARLY AUTHENTICATION AND ENCRYPTION (EAE)

8.1 Introduction

The process of CM initialization is described in [DOCSIS MULPIv4.0]. This section specifies requirements on the CM and the CMTS for securing the CM initialization process. This specification outlines early authentication and encryption (EAE) to DOCSIS CM initialization and places the CMTS in charge of the authentication process on a modem-by-modem basis.

Early authentication functions as a network admission control; only authenticated CMs are allowed to continue their initialization process and may be subsequently admitted to the network. The results of a successful authentication are used for securing subsequent steps in the CM's initialization process.

Early Authentication and Encryption (EAE) refers to the following sequence of processes in their entirety:

1. The authentication of the CM (i.e., the BPI+ Authorization exchanges) following the completion of ranging and before DHCP exchanges (i.e., early authentication);
2. TEK key exchanges for the CM's Primary SAID;
3. Encryption of IP provisioning traffic and the REG-REQ-MP MAC message during CM initialization.

The CMTS MUST support EAE.

The CM MUST support EAE.

8.2 EAE Signaling

When EAE is enabled, the CMTS MUST announce the availability of EAE by using the MDD message. The CM follows procedures specified in [DOCSIS MULPIv4.0] to detect the MDD message.

When using BPI+ V2, the CMTS uses the CMTS BPI+ Enabled Version and Configuration TLV (type 23) in the MDD message to signal EAE for the versions of BPI+ that are enabled on the CMTS. The CM determines whether it is required to perform EAE by detecting the MDD message and inspecting the BPI+ Configuration Bitmask TLV (type 23.2) associated with the version of BPI+ negotiated.

When the CMTS is configured to enable EAE (see Section 8.4, EAE Enforcement), the CMTS MUST include TLV type 23 in the MDD message with the value of the BPI+ Version Number TLV (type 23.1) set to the negotiated BPI+ version number (i.e., 0x01 for BPI+ V1 and 0x02 for BPI+ V2) and with the `EAE-Configuration` bit of the BPI+ Configuration Bitmask TLV (type 23.2) set to one(1). If the CMTS is configured to have EAE disabled, the associated `EAE-Configuration` bit SHOULD be set to zero (0).

When support for EAE is enabled on the CMTS for BPI+ V1, the CMTS MUST also include TLV type 6 in the MDD message with its value set to 1.

When the CMTS is configured to disable EAE for BPI+ V1, the CMTS MUST also include TLV type 6 in the MDD message with its value set to 0.

If the CM detects a valid MDD message before or during initial ranging, and the MDD indicates the EAE is required for the negotiated version of BPI+, then the CM MUST start EAE by sending a {Initiate Authentication} event to the BPI+ Auth State Machine (see Section 7, Baseline Privacy Key Management (BPKM) Protocol) immediately following Ranging completion. If the CM detects no MDD message before or during initial ranging, or the detected MDD message indicates that EAE is disabled for the negotiated version of BPI+, then the CM MUST NOT start EAE. The CM follows procedures specified in [DOCSIS MULPIv4.0] to detect MDD.

The CMTS may omit MDD messages on some downstream channels. EAE only applies to CMs initializing on downstream channels that broadcast the MDD message.

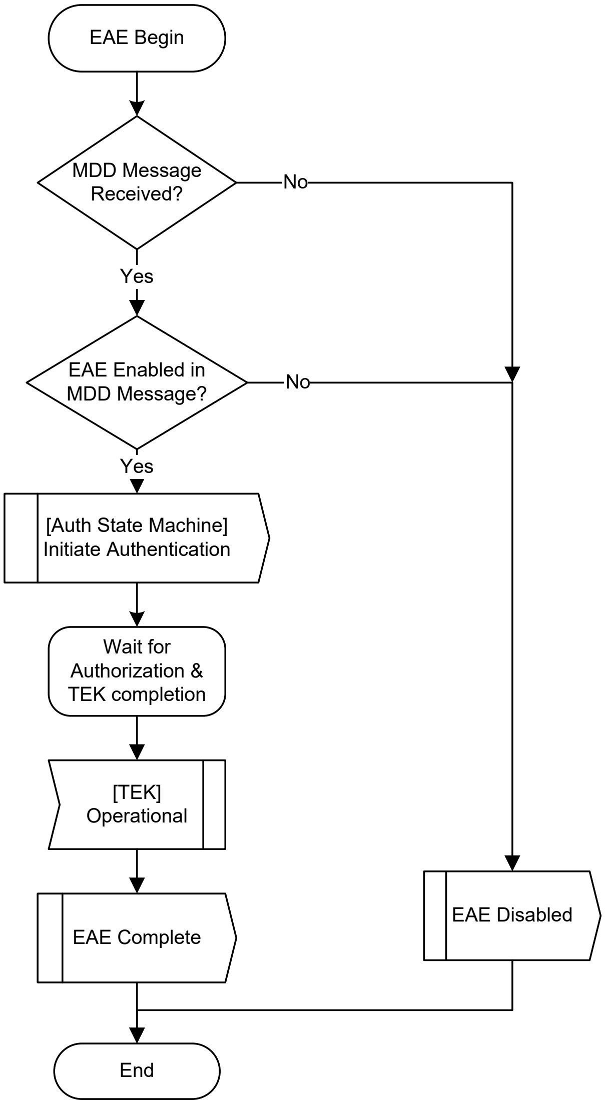
EAE Begin
MDD Message Received?
No
Yes
EAE Enabled in MDD Message?
No
Yes
[Auth State Machine] Initiate Authentication
Wait for Authorization & TEK completion
[TEK] Operational
EAE Complete
EAE Disabled
End

Figure 10 - EAE Signaling Flow Chart for CM

8.3 EAE Encryption

Once a CM has completed EAE Authorization, the CMTS MUST transmit all subsequent unicast DOCSIS data PDUs to that CM that are not otherwise assigned to a Security Association (SA) on the CM's Primary SA, until such time BPI+ is disabled.

A CM which has completed EAE Authorization with a CMTS MUST transmit the following upstream traffic to the CMTS on the CM's Primary Security Association:

- All data PDUs, including DHCP, TOD, and TFTP provisioning traffic; and
- The Registration Request (REG-REQ-MP) MAC management message.

An initializing CM does not encrypt other upstream MAC Management messages to the CMTS (see Section 6.5).

For the purposes of supporting IPv6 managed devices, this specification requires that multicast traffic addressed to the Well-known IPv6 Addresses (see [DOCSIS MULPIv4.0]) be exempted from EAE enforcement. As such, the CMTS MUST NOT encrypt multicast traffic addressed to these Well-known IPv6 Addresses.

8.4 EAE Enforcement

When EAE is enabled, the CMTS uses EAE to perform network admission control by forcing CMs to authenticate before allowing them to proceed with the initialization process. As a result of EAE, a security association is established for the CM's Primary SAID, which protects subsequent provisioning messages (see [DOCSIS MULPIv4.0]).

The CMTS enforces EAE only for CMs that initialize on a downstream channel on which the CMTS is transmitting MDD messages. The CMTS MUST support the following configurable EAE enforcement policies:

Policy 1: No EAE enforcement, i.e., EAE is disabled, and the CMTS does not enforce EAE on any CM.

Policy 2: Ranging-Based EAE Enforcement, i.e., the CMTS enforces EAE on CMs that range with a B-INIT-RNG-REQ MAC message.

Policy 3: Capability-Based EAE Enforcement, i.e., the CMTS enforces EAE on CMs that range with a B-INIT-RNG-REQ MAC message in which the EAE capability flag is set.

Policy 4: Total EAE Enforcement, i.e., the CMTS enforces EAE on all CMs.

The EAE enforcement policies are mutually exclusive. By default, the CMTS MUST enable Ranging-Based EAE Enforcement (Policy 2). Policies 2 and 3 are referred to as Selective EAE Enforcement.

When configured for Selective EAE Enforcement the CMTS does not enforce EAE for DOCSIS 1.1/2.0 CMs since they do not support the B-INIT-RNG-REQ MAC message.

The CMTS enforces EAE on CMs based on the configured EAE enforcement policies. CMs in the EAE Exclusion List (see Section 8.4.5), are always exempted from EAE enforcement.

8.4.1 CMTS and CM Behaviors when EAE is Enabled

When EAE is enabled on a CMTS and a CM performs EAE, the following procedures need to be performed.

- When the ranging process has been successfully completed, the CMTS MUST drop all PDUs (i.e., frames with FC type 00) (see [DOCSIS MULPIv4.0]) from the CM until it has successfully completed EAE. A CM completes EAE when it has received the Key Reply message for the Primary SAID.
- The CMTS MUST use the Primary SA to carry all IP messages involved in the provisioning of the CM (i.e., DHCP, TOD, and TFTP).
- The CM MUST use the Primary SA to carry all IP messages involved in the provisioning of the CM (i.e., DHCP, TOD, and TFTP), and its REG-REQ-MP MAC messages.

8.4.2 EAE Enforcement Determination

This section describes how the CMTS makes EAE enforcement decisions based on its configured policy.

8.4.2.1 Ranging-Based EAE Enforcement

When the CMTS is configured to enable Ranging-Based EAE Enforcement (policy 2), the CMTS enforces EAE on a CM based on the CM's ranging MAC message type, ignoring the EAE capability flag in the B-INIT-RNG-REQ. When the CMTS is configured for EAE enforcement policy 2, it MUST enforce EAE only on CMs that range with B-INIT-RNG-REQ, except for CMs on the EAE Exclusion List (see Section 8.4.5, EAE Exclusion List).

8.4.2.2 Capability-Based EAE Enforcement

When the CMTS is configured to enable Capability-Based EAE Enforcement (policy 3), the CMTS enforces EAE on a CM based on its ranging MAC message type as well as the EAE capability flag in the B-INIT-RNG-REQ.

When the CMTS is configured for policy 3 enforcement, it MUST enforce EAE only on CMs that range with B-INIT-RNG-REQ in which the EAE capability flag is set, except for CMs on the EAE Exclusion List (see Section 8.4.4, CMTS and CM Behavior when EAE is Disabled).

8.4.2.3 Total EAE Enforcement

When the CMTS is configured to enable Total EAE Enforcement (policy 4), the CMTS MUST enforce EAE on all CMs, except for CMs on the EAE Exclusion List (see Section 8.4.4, CMTS and CM Behavior when EAE is Disabled).

8.4.3 EAE Enforcement of DHCP Traffic

When the CMTS is configured to enable EAE with policy 2 or 3 enforcement the CMTS MUST discard DHCP packets from a CM if:

- The Vendor Class Identifier Option (option 60 for DHCPv4 and option 16 for DHCPv6) in the DHCP packets advertise DOCSIS 3.0 or later (see [DOCSIS MULPIv4.0]); and
- The CM has not successfully completed EAE; and
- The CM is not on the EAE Exclusion List (see Section 8.4.5, EAE Exclusion List).

8.4.4 CMTS and CM Behavior when EAE is Disabled

When EAE is disabled, the following procedures need to be performed.

- The CMTS MUST allow a CM to proceed with the Initialization process (see [DOCSIS MULPIv4.0]) without performing EAE;
- The CM MUST NOT initiate EAE after completing initial ranging;
- After completing initial ranging the CM MUST proceed to the next step in the CM initialization process as defined in [DOCSIS MULPIv4.0];
- If the CMTS receives an Authorization Request from a CM following ranging completion, the CMTS SHOULD NOT perform authentication on the CM. The CMTS MUST respond to the Authorization Request with an Authorization Reject message containing the error code 10.

8.4.5 EAE Exclusion List

The CMTS MUST support the capability to exclude individual CMs from EAE enforcement based on their MAC addresses when policy 2, 3, or 4 is enabled on a per-MAC domain basis.

If a CM is on the exclusion list, the following procedures need to be performed:

- The CMTS MUST allow the CM to proceed with the Initialization process (see [DOCSIS MULPIv4.0]) without performing EAE.
- If the CMTS receives an Authorization Request from the CM following ranging completion, the CMTS MUST respond with an Authorization Reject message containing the error code 10.
- If the CM sends an Authorization Request immediately after ranging completion and receives error code 10 in the Authorization Reject message in response, the CM MUST terminate its Authorization state machine and proceed to the next step in its initialization process as described in [DOCSIS MULPIv4.0]. The CM will later initiate Authorization and subsequent TEK key exchanges if it receives a configuration file that enables BPI+ (see Section 7).

8.4.6 Interoperability Issues

A pre-DOCSIS 3.0 CM does not recognize the MDD message and thus will not attempt to perform EAE. The DOCSIS 3.0 CMTS MUST support initialization of pre-DOCSIS 3.0 CMs including operation of the Authorization and TEK state machines following registration as defined in [DOCSIS RFIv2.0].

A DOCSIS 3.0 CM capable of EAE, when deployed against a pre-DOCSIS 3.0 CMTS, determines that EAE is disabled because it does not receive a valid MDD during initial ranging. The process by which a CM detects a valid MDD during initial ranging is described in [DOCSIS MULPIv4.0]. A CM that fails to detect an MDD message proceeds directly to the "Establish IP Connectivity" phase after initial ranging.

8.5 Authentication Reuse

When EAE is enabled, CMs are authenticated immediately following successful ranging. This is early in a CM's initialization process, so a successful authentication can be used to eliminate authentication in subsequent steps during initialization. EAE is also used to secure DHCP, TOD, TFTP, and Registration over the cable network link (see Section 8.3 for more details).

8.6 BPI+ Control by Configuration File

If EAE is enabled for a CM, the CM performs early authentication and establishes its Authorization and TEK state machine for the Primary SAID before receiving its configuration file. If EAE is disabled, the CM receives its configuration file before it is authenticated. In either case, the BPI+ setting in the configuration file (see [DOCSIS MULPIv4.0]) controls all subsequent privacy operations.

8.6.1 EAE Enabled

If EAE is enabled for a CM, its Authorization state machine and the TEK state machine for its Primary SAID are operational by the time the CM receives its configuration file. Depending on the BPI+ settings in the configuration file, the CM's security state machines may continue to operate or may be terminated when the configuration file is processed by the CM.

If the configuration file does not disable BPI+, then the CM's Authorization state machine and its TEK state machine for the Primary SAID MUST continue to operate. If the configuration file enables BPI+ but contains BPI+ settings that differ from the default values listed in Table 57 - Recommended Operational Ranges for BPI+ Configuration Parameters, then the CM MUST update its Auth and TEK state machines with any changed values it receives in the configuration file.

If the Authorization Grace Time value changes, the CM MUST Reauthorize after registration messaging is complete. If the TEK Grace Time value changes, the CM MUST perform a TEK Refresh after registration messaging is complete. If the configuration file contains BG_CFG subtype 3 (Authorization Grace Time, see Annex A) and its value is greater than half the remaining Authorization lifetime, then the CM MUST immediately set the Authorization Grace Time to 1 second less than half the remaining Authorization lifetime. If the configuration file does not contain BPI+ settings, or BPI+ settings are present and have default values listed in Table 57 - Recommended Operational Ranges for BPI+ Configuration Parameters, then the CM's state machines MUST continue to operate as normal EAE (see Section A.2, Parameter Guidelines).

If EAE is enabled for a CM and its configuration file explicitly disables BPI+, then the CM MUST terminate its Authorization state machine and its TEK state machine for the Primary SAID as soon as registration is complete.

8.6.2 EAE Disabled

If EAE is disabled for a CM, then the CM does not initiate any security exchanges with the CMTS until after Registration. The CM's configuration file controls completely whether it initiates any security exchanges with the CMTS.

If EAE is disabled and the CM's configuration file enables BPI+ the following procedures need to be performed:

- The CM MUST send an {Initiate Authentication} event to the Authorization State Machine (see Section 7.1.6, Authorization State Machine) to start CM Authorization process. The Authorization and TEK exchanges between the CM and the CMTS follow the requirements in Section 7.1.1.
- The CM MUST NOT forward traffic from any attached CPE device to the cable network from the time registration completes until after the initialization of Baseline Privacy operations completes for its Primary

SID/SAID. Registration completion is defined in [DOCSIS MULPIv4.0], and initialization of Baseline Privacy operations completion is defined in Section 5.2.1.

- The CMTS maintains the CM's Authorization state and it MUST verify that the CM completed Authorization exchanges with the CMTS before forwarding user data traffic from the CM. The CMTS MUST drop all user data traffic forwarded by the CM until the CMTS verifies that the CM has completed Authorization.

If EAE is disabled and the CM's configuration file disables BPI+, the CM MUST NOT instantiate an Authorization State Machine or start any TEK state machines.

9 SECURE PROVISIONING

9.1 Introduction

The term "secure provisioning" refers to securing the CM provisioning processes. These processes are defined in [DOCSIS MULPIv4.0] and they are DHCP, ToD, and TFTP at the IP layer; and registration at the MAC layer. Secure provisioning plays a critical role in protecting the CMs and the network from attacks, and in preventing service theft. This section places requirements on the CM and CMTS to support secure provisioning.

9.2 Encryption of Provisioning Messages

When EAE is enabled for a CM, all IP provisioning messages are encrypted by the Primary SAID as the payload of DOCSIS packets. Registration Request MAC management messages are also encrypted.

Of special value is the encryption of TFTP messages. The encryption of TFTP packets protects the confidentiality of the contents of CM configuration files downloaded via TFTP. If a configuration file contains sensitive information, EAE should be enabled for that CM.

9.3 Securing DHCP

9.3.1 Securing DHCP on the Cable Network Link

DHCP is a client-server protocol. When EAE is enabled for a CM, security of unicast DHCP messages between the CMTS and the CM is provided by encrypting DHCP packets as they pass across the cable network link. It is assumed in this specification that the path between the DHCP server and the CMTS is secured through mechanisms outside the scope of this specification (see also Section 5).

9.3.2 DHCPv6

CMs support the lightweight DHCPv6 authentication protocol for IPv6 provisioning (see [DOCSIS MULPIv4.0]). As the lightweight DHCPv6 authentication protocol relies on DHCPv6 messages to distribute Reconfigure keys to the CMs, it is essential that DHCPv6 messages be protected. If a CM is provisioned to accept DHCPv6 Reconfigure messages, then enabling EAE adds additional protection for the value of the Reconfigure key.

9.4 TFTP Configuration File Security

9.4.1 Introduction

This section describes requirements intended to secure the CM configuration file download process, and to ensure that the CM does not receive a different level of service than described by the configuration file.

9.4.2 CMTS Security Features for Configuration File Download

The CMTS supports several features intended to secure the download of CM configuration files:

- A capability to prevent the disclosure of the IP address of the configuration file server (TFTP Proxy, see Section 9.4.2.1);
- A capability to enforce that a CM downloads the correct configuration file according to DHCP configurations offered to the CM (Configuration File Name Authorization, see Section 9.4.2.3);
- A capability to verify that a CM registers with settings that match those in the downloaded configuration file (Configuration File Learning, see Section 9.4.2.4).

9.4.2.1 TFTP Proxy

The CMTS MUST implement a TFTP server and a TFTP client compliant with [RFC 1350]. Both the TFTP server and client in the CMTS MUST support TFTP option extension (see [RFC 2347]), TFTP blocksize option (see [RFC

2348]) and TFTP timeout interval option (see [RFC 2349]). The CMTS MUST be capable of acting as the TFTP server for CMs to download configuration files. The CMTS MUST be capable of acting as a TFTP client to download configuration files from TFTP servers in the provisioning system. The CMTS MAY support other file transfer protocol clients for CM configuration file download.

When the CMTS acts as the TFTP server for a CM, and at the same time acts as a TFTP client downloading a configuration file from a TFTP server on behalf of the CM, the CMTS is referred to as a TFTP Proxy. The CMTS MUST support the capability to enable or disable TFTP Proxy. By default, the CMTS MUST enable TFTP Proxy.

9.4.2.2 Protecting TFTP Server Addresses

If TFTP Proxy is enabled on a CMTS and a CM is provisioned in IPv4 mode, then the CMTS MUST ensure that the TFTP Server Address Option and/or the siaddr field in DHCPACK messages sent to the CM is the CMTS's IP address.

If TFTP Proxy is enabled on a CMTS and a CM is provisioned in IPv6 mode, then the CMTS MUST ensure that the CL_OPTION_TFTP_SERVERS suboption of the OPTION_VENDOR_OPTS in Reply messages sent to the CM is the CMTS's IP address.

If TFTP Proxy is enabled and a valid configuration download TFTP request has been received from a CM, the CMTS MUST acquire the configuration file from the configuration server identified in the DHCPACK (DHCPv4), or Reply (DHCPv6) messages relayed to the CM, and download it to the CM.

If TFTP Proxy is enabled on a CMTS, and if the provisioning system uses multiple configuration file servers, then the CMTS SHOULD support a mechanism that uses the multiple TFTP servers. The CMTS SHOULD implement a retry mechanism that synchronizes TFTP retries by the CM and by the CMTS. These mechanisms are not defined by this specification.

9.4.2.3 Configuration File Name Authorization

The CMTS MUST support the capability to maintain a list of authorized DHCP servers.

The CMTS MUST support the capability to learn the name of a CM's configuration file from the DHCP configurations offered to the CM from an authorized DHCP server. The learned configuration file name identifies the configuration file that the CM is authorized to download.

The CMTS MUST support the capability to discard CM TFTP Requests if the name of the configuration file requested by a CM is not identical to the learned name of the configuration file. This capability is referred to as Configuration File Name Authorization. The CMTS MUST enable or disable Configuration File Name Authorization when the TFTP Proxy feature is enabled or disabled, respectively.

9.4.2.4 Configuration File Learning

When TFTP Proxy is enabled on a CMTS, the CMTS downloads configuration files on behalf of CMs, and the CMTS can learn about CMs' configuration files. The CMTS MUST support a capability to learn about the CM's configuration file. This capability is referred to as Configuration File Learning. The CMTS MUST be capable of being configured to enable or disable Configuration File Learning. By default, the CMTS MUST enable Configuration File Learning.

The CMTS MUST support the capability to enforce that a CM's Registration is consistent with what the CMTS has learned about the CM's configuration file.

If TFTP Proxy and Configuration File Learning are both enabled on a CMTS, and the CM's Registration is not consistent with what the CMTS has learned about the CM's configuration file (e.g., based on CMTS MIC calculation, or comparison of parameters used in CMTS MIC calculation), then the CMTS MUST respond with an Authentication Failure in the registration response status field (see [DOCSIS MULPIv4.0]). The CMTS MUST also log an event.

9.4.2.5 TFTP Options for CM's MAC Address and IP Address

When TFTP Proxy is enabled on a CMTS, the client requesting a file from the backend provisioning system is the CMTS rather than the CM. However, some provisioning systems rely on the availability of the CM MAC address and IP address in the request.

In order to allow this information to reach the provisioning system, the CMTS MUST support the MAC address and IP address TFTP options (see Annex B, TFTP Options (Normative)). Enabling support for these options MUST be independently configurable on the CMTS with the default being disabled.

When a CM requests a configuration file and the IP address option is enabled on the CMTS, the CMTS MUST include the CM's IP address in the `netaddr` TFTP option. When a CM requests a configuration file and the MAC address is enabled on the CMTS, the CMTS MUST include the CM's MAC address in the `hwaddr` TFTP option. If a TFTP packet received from a CM already includes these options, the CMTS MUST discard those options and include only the enabled TFTP options with source address values from the received packet. When either the IP address or MAC address option is enabled, the CMTS MUST NOT cache configuration files locally.

9.5 Securing REG-REQ-MP Messages

Encryption of packet data carried in DOCSIS frames provides confidentiality of CM configuration files. A subset of the parameters contained in configuration files is communicated to the CMTS in REG-REQ-MP messages (see [DOCSIS MULPIv4.0]). To maintain confidentially of these parameters, when EAE is enabled the CM MUST encrypt the REG-REQ-MP message (see Section 6.5, Registration Request (REG-REQ-MP) MAC Management Messages). The key for encrypting the REG-REQ-MP message is the Primary SA TEK. If EAE is enabled and the configuration file disables BPI+, the CM MUST complete registration before terminating the authorization and TEK state machines.

9.6 Source Address Verification

The CMTS is responsible only for forwarding CPE packets that contain legitimate addresses. This section imposes on the CMTS requirements designed to permit an operator to ensure that CPEs located behind CMs cannot successfully spoof addresses in order to obtain unauthorized access to services or to disrupt services to others. These CMTS requirements are referred to as Source Address Verification (SAV).

A design goal of the SAV feature is that it applies in deployment scenarios where CPEs are directly connected to a CM and where CPEs are behind a router that is connected to a CM. The router may be embedded with the CM or standalone.

The CMTS MUST be capable of being configured to enable and disable SAV. By default, the CMTS MUST enable SAV. When the SAV feature is enabled, the CMTS MUST drop any received upstream packets whose IP source address has not been assigned by the operator. This includes packets whose source IP address is an IP address that has been assigned to another device. When multiple devices are assigned the same IP address, it is left to CMTS vendor implementation to determine which packet to drop when upstream packets from different devices with the same assigned source IP address are received. Source IP addresses are considered assigned by the operator when they are provisioned via DHCP messaging or identified by parameters in the configuration file.

The CM configuration file can contain the following TLV encodings (see [DOCSIS MULPIv4.0]) that are used to indicate IP addresses that have been assigned by the operator, but are not issued by a DHCP server:

1. An SAV Prefix Group ID Encoding that identifies a list of prefixes configured at the CMTS; or
2. A Static SAV Prefix Encoding that statically defines an IP prefix authorized for source IP addresses of upstream traffic from the CM.

A valid CM configuration file has a single SAV Group Encoding and may have one or more Static SAV Prefix Encodings.

The CMTS MUST consider all upstream IP packets on all SIDs assigned to a CM as containing a source IP address assigned by the operator when the source IP address:

1. Matches the Subscriber Management CPE IPv4 or IPv6 Encoding (see [DOCSIS MULPIv4.0]) signaled for that CM in a REG-REQ-MP; or
2. Matches a prefix in the CMTS prefix list referenced by the SAV Group ID Encoding (see [DOCSIS MULPIv4.0]) signaled for that CM in a REG-REQ-MP; or
3. Matches a Static SAV Prefix Encoding signaled for that CM in a REG-REQ-MP, or
4. Was learned from an authorized DHCP server. Examples include:
   a) The CMTS extracts assigned IP address information from DHCPv4 DHCPACK message sent in response to an upstream DHCPv4 DHCPREQUEST, or from a DHCPv6 Reply message sent in response to an upstream DHCPv6 Request received from a CM's SID;
   b) Use of a CMTS initiated DHCPv4 DHCPLEASEQUERY or DHCPv6 Leasequery response that verifies the assigned IP address for a host source MAC address;
   c) The CMTS extracts assigned IPv6 prefix from an IPv6 prefix delegation option.

The DHCP Leasequery protocols have been specified to facilitate the exchange of data between a CMTS and DHCP server. The CMTS MAY implement the following standards to support interactions with provider DHCP server(s) related to SAV: the DHCP Leasequery protocol which is specified in [RFC4388]; the DHCPv6 Leasequery protocol and related standards which are specified in [RFC5007] and [RFC4994]. The CMTS MAY implement other mechanisms for determining whether IP addresses have been assigned by the operator, e.g., when the source IP address is within an IP subnet authorized as routed downstream to a next hop router reached through the CM.

The CMTS MUST be capable of being configured to enable and disable the use of the SAV Group ID Encoding and Static SAV Prefix Encoding for identification of operator assigned IP addresses.

When Source Address Verification is enabled, a routing CMTS MUST respond in proxy to an upstream address resolution request (IPv4 ARP Request or IPv6 Neighbor Solicitation) for downstream target IP addresses that it has verified instead of reflecting the request downstream.

Whether or not SAV is enabled, the CMTS MUST discard upstream IP packets received on a SID assigned to an initializing CM and all upstream IP packets on all SIDs of the CM after it has finished the initialization process when the source IP address does not match the IP address assigned to the CM by an authorized DHCP server. An initializing CM is one that has not yet completed registration.

The CMTS provides per-CM statistics of the number of packets discarded due to SAV failure.

9.7 Address Resolution Security Considerations

Address Resolution Protocol (ARP) (see [RFC 826]) is a protocol for dynamically mapping an IP address to the corresponding link layer address (e.g., IEEE Ethernet MAC address) in the local network. Neighbor Discovery (ND) is an IPv6 protocol (see [RFC 4861]) that provides the same function. A routing CMTS uses ARP or ND for IP address resolution of downstream IP host addresses in a directly connected downstream IP subnet. Downstream IP subnets include the IP addresses for CMs, eSAFEs, and CPE hosts directly connected to the network segment to which a CM's external CPE interface attaches.

ARP is a broadcast protocol. ARP requests are broadcast to, and received and processed by, all devices on the network segment. An IPv6 Neighbor Solicitation message for address resolution is a multicast packet. For purposes of this section, an "address resolution request" packet is defined as an IPv4 ARP request or IPv6 Neighbor Solicitation (NS) message. An "address resolution response" packet is an IPv4 ARP reply or an IPv6 Neighbor Advertisement (NA) message. This section describes CMTS requirements concerning the monitoring and limiting of address resolution request messages.

Because the downstream and upstream are physically separate media in DOCSIS technology, CMs cannot directly perform address resolution with other CMs on the cable plant. As a result, the CMTS is involved in the MAC-to-IP address resolution process, either by echoing resolution request / response packets between downstream and upstream, by acting in proxy for the address resolution target, or by bridging the resolution request/response packets between the cable interfaces and the NSI interfaces.

If the CMTS knows the Ethernet MAC address corresponding to a host IP address, the host IP address is said to be "resolved" at the CMTS; otherwise, the host IP address is said to be "unresolved." When a routing CMTS attempts to forward an IP packet (upstream or downstream) to an unresolved downstream host IP address, the CMTS MAY originate a downstream address resolution request for that host IP address. When a CMTS receives an upstream address resolution request to an unresolved target, it MAY reflect that request downstream. The [DOCSIS MULPIv4.0] specification describes requirements for downstream forwarding of broadcast ARP requests and multicast NS requests.

Because ARP is a broadcast protocol and downstream address resolution (ARP or ND) consumes CMTS resources for pending address resolution requests, a routing CMTS is susceptible to a denial of service attack that attempts to exhaust the CMTS's resources for pending address resolutions. Furthermore, ARP "storms" can occur when large numbers of ARP requests or unresolved IP destination packets are received by the CMTS, impairing the ability of the CMTS to process legitimate traffic. ARP storms may also be caused by ill-configured or malfunctioning customer equipment and computer systems.

A characteristic of a CPE host infected with a virus is that it attempts to discover other potential victims by originating ARP / NS requests to many other hosts in its same IP subnet. During virus attacks, hundreds of packets per second to unresolved host IP addresses can be received by the CMTS. Although the Neighbor Solicitation message is a solicited-node multicast address instead of a broadcast address, the occurrence of a high rate of NS messages upstream from a CM is a likely indicator that a CPE host connected to the CM is infected with a virus that is attempting to discover other hosts on the same IPv6 link layer subnet.

This section defines CMTS requirements for mitigating ARP storms, and to assist with the discovery of which CMs attach to CPE hosts that may be infected by computer viruses.

The Source Address Verification (SAV) feature, when enabled, mitigates address resolution denial of service by requiring a routing CMTS to respond in proxy to upstream address resolution requests instead of reflecting them down stream.

The CMTS MUST support the capability to limit the rate at which it transmits downstream address resolution requests. This requirement applies to downstream address resolution requests generated either when reflecting an upstream request or to resolve the Ethernet MAC address of an IP packet to be forwarded downstream. The configuration of downstream address resolution request limiting is CMTS vendor specific.

The CMTS MUST implement a management object for each CM that accumulates the count of upstream address resolution request packets received on SIDs assigned to the CM. The upstream address resolution request packet count includes the following:

- Upstream IPv4 ARP Requests;
- Upstream IPv6 Neighbor Solicitation Requests;
- (For routing CMTSs) Upstream IPv4 or IPv6 packets to unresolved destinations in locally connected downstream subnets.

The Upstream Address Resolution Requests counter is intended to be analyzed by MSO management processes that obtain the counter via SNMP or IPDR. CMs with high rates of upstream address resolution requests are likely to have CPEs infected with viruses attempting to locate other victims.

10 USING CRYPTOGRAPHIC KEYS

10.1 CMTS

The CMTS's first receipt of an Authorization Request message from an unauthorized CM initiates the activation of a new Authorization Key (AK), which the CMTS returns to the CM in an Authorization Reply message (see Section 7). This AK will remain active until it expires according to its predefined lifetime, Authorization Key Lifetime, which is a CMTS system configuration parameter (see Annex A).

The CMTS uses keying material derived from the CM's Authorization Key for:

- Verifying the HMAC-Digest in Key Requests received from the CM;
- Encrypting (EDE mode two-key 3DES) the TEK in the Key Replies that it sends to the CM;
- Calculating the HMAC-Digests in Key Replies, Key Rejects, and TEK Invalids sent to the CM.

See Section 7 for TEK messaging (Key Requests, Key Replies, Key Rejects, etc.) details.

The CMTS MUST be prepared to send an AK upon request. The CMTS MUST be able to support two simultaneously active AKs for each client CM.

If the CMTS holds two active Authorization Keys for a CM, it responds to Authorization Requests with the newer of the two active keys. If the CMTS holds a single active Authorization Key, a received Authorization Request will trigger the activation of a new AK, as described below.

An Authorization Key "transition period" begins when the CMTS receives an Authorization Request from a CM and the CMTS has a single active AK for that CM. In response to this Authorization Request, the CMTS activates a second AK, which it returns to the requesting CM in an Authorization Reply. The CMTS MUST set the active lifetime of this second AK to be the remaining lifetime of the first AK plus the predefined Authorization Key Lifetime. The key transition period ends with the expiration of the older key. This is depicted in the top half of Figure 11.

The Authorization Key lifetime that the CMTS reports in an Authorization reply MUST be set to the remaining lifetimes of the AKs at the time the reply message is sent.

If a CM fails to reauthorize before the expiration of its most recently acquired AK, the CMTS will hold no active Authorization keys for the CM and will consider the CM unauthorized. A CMTS MUST deactivate all TEKs associated with an unauthorized CM.

The CMTS tracks the lifetime of its Authorization Keys; the CMTS MUST immediately deactivate a key once it has expired.

A CMTS MUST use a CM's active AK(s) to verify the HMAC-Digest in Key Requests received from the CM. If a CMTS receives a Key Request while in an AK transition period, and the AK Key Sequence Number indicates that the Request was authenticated with the newer of the two AKs, the CMTS MUST recognize this as an implicit acknowledgment that the CM has obtained the newer of the CM's two active AKs.

The CMTS MUST use an active AK when calculating HMAC-Digests in Key Replies, Key Rejects and TEK Invalids, and when encrypting the TEK in Key Replies. When sending Key Replies, Key Rejects or TEK Invalids within a key transition period and the newer key has been implicitly acknowledged, the CMTS MUST use the newer of the two actives. If the newer key has not been implicitly acknowledged, the CMTS MUST use the older of the two active AKs.

Figure 11 illustrates the CMTS's use of AKs.

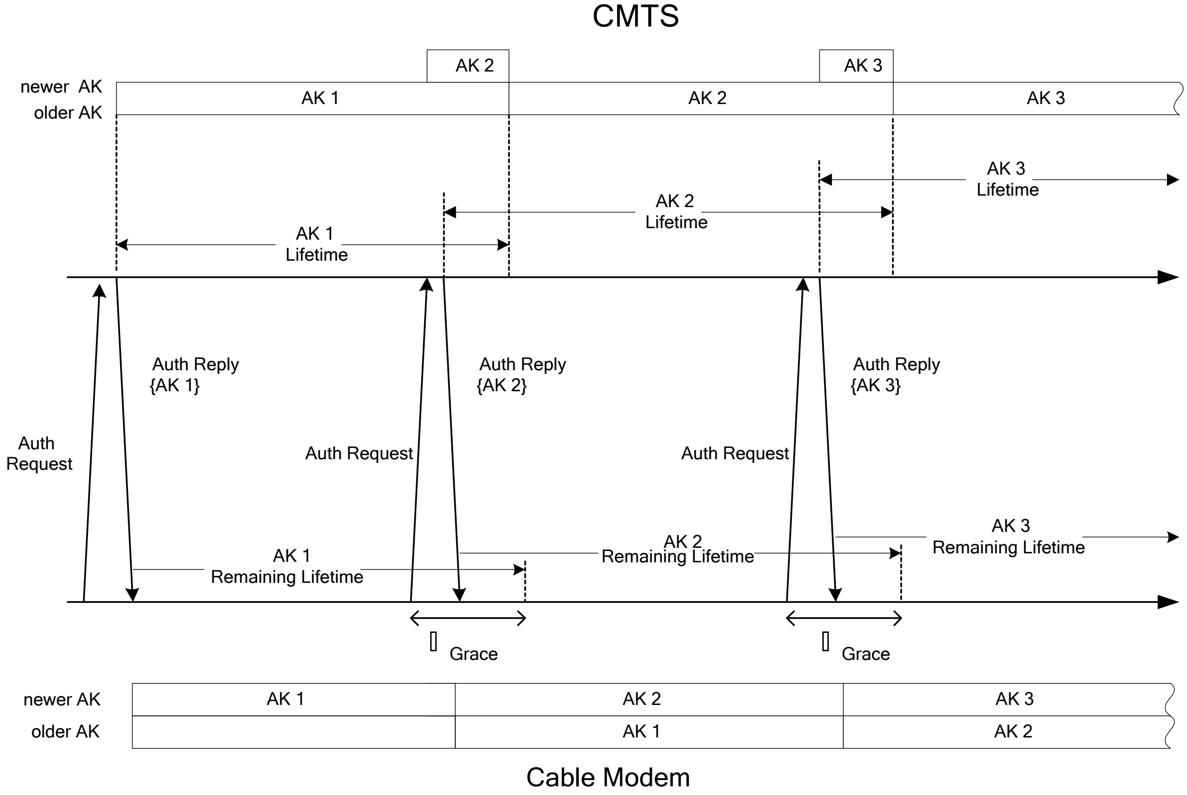

Figure 11 - Authorization Key Management in CMTS and CM

The CMTS MUST be capable of maintaining two sets of active traffic encryption keys (and their associated CBC initialization vectors) per SAID. These correspond to two successive generations of keying material and have overlapping lifetimes. The CMTS MUST make the newer TEK have a key sequence number one greater than that of the older TEK (modulo 16). Each TEK becomes active halfway through the lifetime of its predecessor and expires halfway through the lifetime of its successor. Once a TEK expires, the TEK becomes inactive and the CMTS MUST NOT use that TEK.

For each of its SAIDs, the CMTS transitions between active TEKs according to the following rules:

- For encrypting downstream traffic, the CMTS MUST use the oldest available active TEK;
- For decryption of upstream traffic, a transition period is defined that begins once the CMTS has sent the newer TEK to a CM within a Key Reply Message. The upstream transition period begins from the time the CMTS sends the newer TEK in a Key Reply Message and concludes once the older TEK expires. While in the transition period, the CMTS MUST be able to decrypt upstream traffic with whichever of the two active TEKs was used to encrypt it.

In other words, the CMTS encrypts with a given TEK for only the second half of that TEK's total lifetime; the CMTS is able, however, to decrypt with a given TEK for that TEK's entire lifetime.

The KEY_SEQ field in the Baseline Privacy EH element identifies which of the two active TEKs was used to encrypt the upstream frame's packet data. The TOGGLE bit in the Privacy EH element (which is required to be equal to the least significant bit of the KEY_SEQ field) can be used by the CMTS to identify the encrypting TEK. Figure 12 illustrates how a CMTS manages TEKs. The Key Replies sent by a CMTS contain TEK parameters (the TEK itself, a key lifetime, a key sequence number and a CBC IV) for the two active TEKs. The key lifetimes that a CMTS reports in a Key Reply MUST be set to the remaining lifetimes of these TEKs at the time that the Key Reply message is sent.

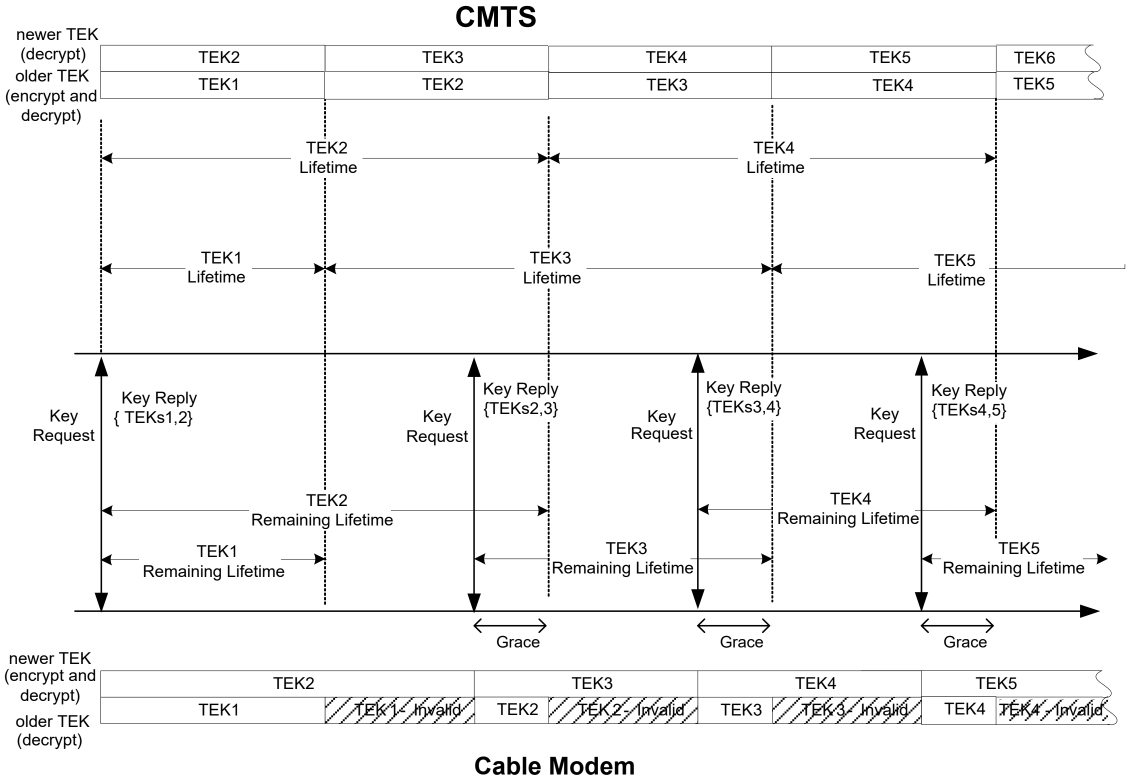

Figure 12 - TEK Management in CMTS and CM

10.2 Cable Modem

The CM is responsible for maintaining two active Authorization Keys. A CM MUST be able to use any active AK. AKs have a limited lifetime and are periodically refreshed. A CM refreshes its Authorization Key by sending an Authorization Request to the CMTS. The Authorization state machine (Section 7.1.6) manages the scheduling of Authorization Requests for refreshing AKs.

A CM's Authorization state machine schedules the beginning of reauthorization a configurable length of time (the Authorization Grace Time) before the CM's most-recently-acquired AK is scheduled to expire. The Authorization Grace Time is configured to provide a CM with an authorization retry period that is sufficiently long to allow for system delays and to provide adequate time for the CM to complete an Authorization exchange before the expiration of its most current AK.

The CM MUST use the most recently acquired Authorization Key when calculating the HMAC-Digests attached to Key Requests. The CM MUST be able to use either of its two most recently acquired AKs to authenticate Key Replies, Key Rejects and TEK Invalids, and to decrypt a Key Reply's encrypted TEK. The CM MUST use the AK Key Sequence Number to determine which of the two AKs to use.

The lower half of Figure 11 illustrates a CM's maintenance and use of its Authorization Keys.

A CM MUST be capable of maintaining two sets of traffic keying material per authorized SAID. Through operation of its TEK state machines, a CM attempts to maintain an SAID's two most recent sets of traffic keying material.

For each of its authorized SAIDs, the CM:

- MUST use the newer of its two TEKs to encrypt newly received upstream traffic.
- MUST use an unexpired TEK for traffic already queued for transmission.
- MUST be able to decrypt downstream traffic encrypted with either active TEK.

10.3 Authentication of Dynamic Service Requests

10.3.1 CM

If BPI+ is enabled, the CM MUST include HMAC-Digests in the following MAC management messages:

- DSA-REQ, DSA-RSP, DSA-ACK;
- DSC-REQ, DSC-RSP, DSC-ACK;
- DSD-REQ;
- DCC-RSP;
- DBC-RSP.

These HMAC-Digests are keyed with the message authentication keys derived from the Authorization Key. The CM MUST use the current message authentication keys when generating and validating the HMAC-Digests contained in the above messages.

10.3.2 CMTS

If BPI+ is enabled for a CM, the CMTS MUST include HMAC-Digests in the following MAC management messages sent to that CM:

- DSA-REQ, DSA-RSP, DSA-ACK;
- DSC-REQ, DSC-RSP, DSC-ACK;
- DSD-REQ;
- DCC-REQ, DCC-ACK;
- DBC-REQ, DBC-ACK.

These HMAC-Digests are keyed with the message authentication keys derived from the Authorization Key. The CMTS MUST use the current message authentication keys when generating and validating the HMAC-Digests contained in the above messages.

11 CRYPTOGRAPHIC METHODS

This section specifies cryptographic algorithms and key sizes.

11.1 Packet Data Encryption

The CMTS MUST use the CBC mode (see [NIST-800-38A]) of either the Data Encryption Standard (DES) algorithm (see [FIPS 46-3]) or the Advanced Encryption Standard (AES) algorithm (see [FIPS 197]) to encrypt the Packet Data field, RF MAC PDU Frames.

The CM MUST use the CBC mode (see [NIST-800-38A]) of either the Data Encryption Standard (DES) algorithm (see [FIPS 46-3]), or the Advanced Encryption Standard (AES) algorithm (see [FIPS 197]) to encrypt the Packet Data field, RF MAC PDU Frames, and the Fragmentation Payload and Fragmentation CRC Fields in MAC Fragmentation Frames.

The CM MUST support one-hundred twenty-eight (128)-bit AES (i.e., a 128-bit key) with a one-hundred twenty-eight (128)-bit block.

The CM MAY support two-hundred fifty-six (256)-bit AES (i.e., a 256-bit key) with a one-hundred twenty-eight (128)-bit block.

The CM MUST support fifty-six (56)-bit DES. The CM MAY support forty (40)-bit DES.

The CMTS MUST support one-hundred twenty-eight (128)-bit AES (i.e., 128-bit key) with a one-hundred twenty-eight (128)-bit block.

The CMTS MAY support two-hundred fifty-six (256)-bit AES (i.e., a 256-bit key) with a one-hundred twenty-eight (128)-bit block.

The CMTS MUST support fifty-six (56)-bit DES.

The CMTS MAY support forty (40)-bit DES.

Forty (40)-bit DES is identical to fifty-six (56)-bit DES, with the exception that sixteen (16) bits of the fifty-six (56)-bit DES key are set to known, fixed values.

If a CM is running the optional forty (40)-bit DES, the CM MUST mask off (to zero) the sixteen (16) left-most bits of any fifty-six (56)-bit DES key prior to running encryption/decryption operations.

If a CMTS is running the optional forty (40)-bit DES, the CMTS MUST mask off (to zero) the sixteen (16) left-most bits of any fifty-six (56)-bit DES key prior to running encryption/decryption operations.

NOTE: The masked bits are the sixteen (16) left-most bits that would be present after the removal of every eighth bit from the sixty-four (64)-bit TEK (i.e., the so-called parity bits). DOCSIS 1.1 or 2.0 and fifty-six (56)-bit DOCSIS 2.0 hardware running BPI+ may implement forty (40)-bit DES key masking in software.

The CMTS MUST initialize CBC mode with the initialization vector that is provided in the CMTS's Key Reply.

The CMTS MUST perform chaining block-to-block within a frame.

The CMTS MUST reinitialize chaining with each frame.

The CM MUST initialize CBC mode with the initialization vector that is provided in the CMTS's Key Reply.

The CM MUST perform chaining block-to-block within a frame.

The CM MUST reinitialize chaining with each frame.

The CMTS MUST use residual termination block processing, as defined below, to encrypt the final block of plain text when the final block is less than the block length defined for the encryption algorithm.

The CM MUST use residual termination block processing, as defined below, to encrypt the final block of plain text when the final block is less than the block length defined for the encryption algorithm.

Given a final block having n bits, where n is less than the defined block length for the encryption algorithm:

- The next-to-last ciphertext block is encrypted a second time, using the ECB mode of the encryption algorithm, and the left most *n* bits of the result are XOR'd with the final *n* bits of the payload to generate the short final cipher block. In order for the receiver to decrypt the short final cipher block, the receiver encrypts the next-to-last ciphertext block using the ECB mode of the encryption algorithm, and XORs the left-most *n* bits with the short final cipher block in order to recover the short final cleartext block (see [SCTE 52] for more details).
- In the special case when the frame's to-be-encrypted plaintext is less than the length of the block, the initialization vector is encrypted, and the left-most *n* bits of the resulting ciphertext corresponding to the number of bits of the payload are XORed with the *n* bits of the payload to generate the short cipher block.

NOTE: This method of encrypting short payloads is vulnerable to attack: XORing two sets of ciphertext encrypted in the above manner under the same set of keying material will yield the XOR of the corresponding sets of plaintext. In the case of PDU frames, however, this is not an issue since all frames carrying protected user data will contain at least 20 bytes of IP header. In the case of Fragmentation Frames, short frames are possible, and a few octets may be exposed by this XOR attack.

11.2 Encryption of the TEK

The CMTS encrypts the value fields of the TEK in Key Reply messages. This field is encrypted using two-key 3DES in the encrypt-decrypt-encrypt (EDE) mode.

Encryption: $C = E_{k1}[Dk2[E_{k1}[P]]]$

Decryption: $P = D_{k1}[E_{k2}[D_{k1}[C]]]$

P = Plaintext TEK

C = Ciphertext TEK

k1 = left-most 64 bits of the 128-bit KEK

k2 = right-most 64 bits of the 128-bit KEK

E[ ] = 56-bit DES ECB mode encryption

D[ ] = 56-bit DES ECB decryption

Section 11.4 describes how the KEK is derived from the Authorization key.

11.3 HMAC-Digest Algorithm

When creating or verifying the `HMAC-Digest` attribute, the CMTS MUST use the HMAC message authentication method (see [RFC 2104]) with the SHA-1 hash algorithm (see [FIPS 180-4]).

When creating or verifying the `HMAC-Digest` attribute, the CM MUST use the HMAC message authentication method (see [RFC 2104]) with the SHA-1 hash algorithm (see [FIPS 180-4]).

Upstream and downstream message authentication keys are derived from the Authorization Key (see Section 11.4 for details).

11.4 TEKs, KEKs, and Message Authentication Keys

The CMTS generates Authorization Keys, TEKs and IVs.

The CMTS MUST use a random or pseudo-random number generator to generate Authorization Keys, TEKs and IVs.

It is recommended the CMTS follow practices recommended in [RFC 4086] for generating random numbers for use within cryptographic systems. [FIPS 46-3] defines DES keys as 8-octet (64-bit) quantities in which the seven most significant bits (i.e., seven left-most bits) of each octet are the independent bits of a DES key, and the least significant bit (i.e., right-most bit) of each octet is a parity bit computed on the preceding seven independent bits and adjusted so that the octet has odd parity.

The CM MUST ignore the value of the least significant bit of each octet in DES keys.

It is not necessary for the CMTS to calculate parity bits in generated DES keys.

The keying material for two-key 3DES consists of two distinct DES keys.

A key encryption key (KEK) and two message authentication keys (HMAC_KEY_U for authenticating upstream Key Request messages, HMAC_KEY_D for authenticating downstream Key Reply, Key Reject and TEK Invalid messages) are derived from a common Authorization Key. The CMTS MUST derive these keys, as shown below. The CM MUST derive these keys as shown below:

KEK = Truncate (SHA-1(K_PAD | AUTH_KEY), 128)

HMAC_KEY_U = SHA-1(H_PAD_U | AUTH_KEY)

HMAC_KEY_D = SHA-1(H_PAD_D | AUTH_KEY)

Where:

SHA-1(x | y) denotes the result of applying the SHA-1 function to the concatenated bit strings x and y;

Truncate(x,n) denotes the result of truncating x to its left-most n bits;

and K_PAD, H_PAD_U and H_PAD_D are 512-bit strings:

K_PAD = 0x53 repeated 63 times;

H_PAD_U = 0x5C repeated 63 times;

H_PAD_D = 0x3A repeated 63 times; and

AUTH_KEY is the Authorization Key.

11.5 Secure Authorization Key Distribution

The key exchange mechanism for delivering the Authorization Key is performed differently in BPI+ V1 and BPI+ V2.

Section 11.5.1 provides a description of the encryption process for Authorization Keys in BPI+ V1.

Section 11.5.2 provides a description for Elliptic-Curves Diffie-Hellman key exchange that is used in BPI+ V2.

11.5.1 Public-Key Encryption of Authorization Key in BPI+ V1

The CMTS MUST encrypt authorization keys in Authorization Reply messages with the CM's public key.

The CM public key MUST be suitable for use with the RSA algorithm (see [RSA3]).

The CM's RSA key MUST have a public exponent of F_4 (65537 decimal, 0x010001).

The CM's RSA key MUST have a modulus length of 1024 bits for the legacy PKI certificate and 2048 bits for the new PKI certificate.

The CMTS MUST use the RSAES-OAEP encryption scheme defined in [RSA3].

When performing this encryption, the CMTS MUST use:

1. SHA-1 for the hash function;
2. MGF1 with SHA-1 for the mask-generation function; and
3. The empty string for the encoding parameter string.

In order to interoperate with earlier versions of this specification, the CMTS MUST support 768-bit and 1024-bit key moduli.

11.5.2 Authorization Key derivation in BPI+ V2

As described in Section 7.2.2.35, the Authorization Key derivation in BPI+ V2 uses the ephemeral Elliptic-Curves Diffie-Hellman algorithm to deliver Perfect Forward Secrecy (PFS).

In particular, differently from BPI+ V1 where the CMTS generates the Authorization Key and then encrypts it for the CM before sending it to the CM in the Authorization Reply message, BPI+ V2 requires both parties (i.e., the CM and the CMTS) to participate in the key derivation process by exchanging their own ECDHE public parameters during the authorization process. Each party derives the Authorization Key by combining the other party's public key exchange parameters, its own public parameters, and its own private ones.

The CM MUST follow the procedures described in Section 7.4.2 of [RFC 8446] when deriving the Authorization Key from the provided `Key-Exchange-Share` parameters.

The CMTS MUST follow the procedures described in Section 7.4.2 of [RFC 8446] when deriving the Authorization Key from the provided `Key-Exchange-Share` parameters.

11.6 Digital Signatures

The default hash algorithm for signatures is SHA-256. SHA-384 and SHA-512 can also be used when calculating digital signatures (see [FIPS 180-4]).

11.7 The MMH-MIC

In this section the MMH Function and the MMH MIC are described. The MMH MIC is included in the CM configuration file to verify the integrity of various encodings (see [DOCSIS MULPIv4.0]).

11.7.1 The MMH Function

The Multilinear Modular Hash (MMH) function described below is a variant of the MMH function described in [MMH]. Some of the computations described below use signed arithmetic whereas the computations in [MMH] use unsigned arithmetic. The signed arithmetic variant described here was selected for its computational efficiency. All of the properties shown for the MMH function in [MMH] continue to hold for the signed variant.

The MMH function has three parameters: the word size, the number of words of input, and the number of words of output. MMH[ω, σ, t] specifies the hash function with word size ω, σ input words and t output words. For DOCSIS 4.0 technology, the word size is fixed to 16 bits: ω =16. The number of output words is fixed at 4 (i.e., 8 octets): $t = 4$. Thus, DOCSIS 4.0 technology uses MMH[16, σ, 4]. The following sections first describe the calculation MMH[16, σ, 1], followed by the method to extend this to MMH[16, σ, 4].

11.7.1.1 MMH[16, σ, 1]

For the remainder of this section, MMH[16, σ, 1] is denoted by the symbol $\mathcal{H}^1$. In addition to σ words of input, $\mathcal{H}^1$ also takes as input a key of σ words. When $\mathcal{H}^1$ is used in computing the MMH-MAC, the key is denoted by κ and the ith word of the key by κ_i:

$$\kappa = \kappa_1, \kappa_2, ..., \kappa_\sigma$$

Likewise, the input message is denoted by M and the ith word of the input message by M_i:

$$M = M_1, M_2, ..., M_\sigma$$

To describe $\mathcal{H}^1$, the following definitions are needed:

For any even positive integer n, S_n is the set of n integers:

$$S_n = \{-n/2, ..., 0, ..., (n/2) - 1\}$$

For example,

$$S_{2^{16}} = \left\{-2^{15},...,0,...,2^{15}-1\right\}$$

is the set of signed 16-bit integers.

For any integer *z*, *z σ* mod *n* is the unique element *ω* of S_n such that $z \equiv \omega$, modulo *n*.

For example, if *z* is a 32-bit signed integer in 32-bit twos-complement representation, then $z \ \sigma \bmod 2^{16}$ can be computed by taking the 16 least-significant bits of *z* and interpreting those bits in 16-bit twos-complement representation.

For any positive integer *q*, the set of *q* integers $\{0,1,...,q-1\}$ is denoted Z_q.

As described above, $\mathcal{H}^1$ takes as input a key of *σ* words (each of length 16 bits). Each of the *σ* words of the key is interpreted as a 16-bit signed integer, i.e., an element of $S_{2^{16}}$.

$\mathcal{H}^1$ also takes an input message of *σ* words. Like the key, each of the *σ* words is interpreted as a 16-bit signed integer, i.e., an element of $S_{2^{16}}$.

The output of $\mathcal{H}^1$ is a single unsigned 16-bit integer, i.e., an element of $Z_{2^{16}}$. In other words, the range of $\mathcal{H}^1$ is $S_{2^{16}}{}^{\sigma} \times S_{2^{16}}{}^{\sigma}$ and the domain is $Z_{2^{16}}$.

$\mathcal{H}^1$ is defined by the following series of steps. Each step is discussed in further detail below:

For *κ*, $M \in S_{2^{16}}{}^{\sigma}$,

Define $\mathcal{H}_1$ as $\mathcal{H}_1(\kappa, M) = \sum_{i=1}^{\sigma} \kappa_i \cdot M_i \sigma \bmod 2^{32}$.

Define $\mathcal{H}_2$ as $\mathcal{H}_2(\kappa, M) = \mathcal{H}_1(\kappa, M) \bmod p$, where *p* is the prime number $p = 2^{16} + 1$.

Define $\mathcal{H}^1$ as $\mathcal{H}^1(\kappa, M) = \mathcal{H}_2(\kappa, M) \bmod 2^{16}$.

Equivalently,

$$\mathcal{H}^1(\kappa, M) = \left(\left(\left(\sum_{i=1}^{\sigma} \kappa_i, M_i\right) \sigma \bmod 2^{32}\right) \bmod p\right) \bmod 2^{16}$$

Step 1. $\mathcal{H}_1(\kappa, M)$ is the inner product of two vectors each of *σ* 16-bit signed integers. The result of the inner product is taken $\sigma \bmod 2^{32}$ to yield an element of $S_{2^{32}}$. That is, if the inner product is in twos-complement representation of 32 or more bits, the 32 least significant bits are retained, and the resulting integer is interpreted in 32-bit twos-complement representation.

Step 2. This step consists of taking an element x of $S_{2^{32}}$ and reducing it $\bmod p$ to yield an element of Z_p. If x is represented in 32-bit twos-complement notation then this reduction can be accomplished as follows. Let a be the unsigned integer given by the 16 most significant bits of x. Let b be the unsigned integer given by the 16 least significant bits of x. There are two cases depending upon whether x is negative.

Case 1: If $x \geq 0$, then $x = 2^{16}a + b$, where $a \in \{0,...,2^{15} - 1\}$ and $b \in \{0,...,2^{16} - 1\}$.

From the modular equation:

$$2^{16}a + b \equiv \left(2^{16}a + b - a\left(2^{16} + 1\right)\right)\bmod\left(2^{16} + 1\right)$$

it follows that $x \equiv b - a$, $\bmod\, p$. The quantity $b - a$ is in the range $\{-2^{15} + 1,...,2^{16} - 1\}$. Therefore, if $(b - a) > 0$, then $x \bmod p = b - a$. If $b - a < 0$, then $x \bmod p = b - a + p$.

Case 2: If $x < 0$, then $x = 2^{16}a + b - 2^{32}$, where $a \in \{2^{15},...,2^{16} - 1\}$ and $b \in \{0,...,2^{16} - 1\}$.

From the modular equation:

$$2^{16}a + b - 2^{32} \equiv \left(b + 2^{16}a - a\left(2^{16} + 1\right) - 2^{32} + 2^{16}\left(2^{16} + 1\right)\right)\bmod\left(2^{16} + 1\right),$$

it follows that $x \equiv \left(b - a + 2^{16}\right)\bmod p$. The range of the quantity $b - a + 2^{16}$ is given by:

$$1 \leq b - a + 2^{16} \leq 2^{17} - 2^{15} - 1 \leq 2p - 1.$$

Therefore, if $b - a + 2^{16} < p$, then $x \bmod p = b - a + 2^{16}$. If $b - a + 2^{16} \geq p$, then $x \bmod p = b - a + 2^{16} - p$

Step 3. This step takes an element of Z_p and reduces it modulo 2^{16}. This is equivalent to taking the 16 least significant bits.

11.7.1.2 MMH[16, σ, n]

This section describes the MMH function with an output length of *n* words.

For convenience, let $\mathcal{H}^n = MMH[16,\sigma,n]$. $\mathcal{H}^n$ takes a key of $\sigma + n - 1$ words. Let $\kappa = \kappa_1,...,\kappa_{\sigma+1}$.

Furthermore, define $\kappa^{(q)}$ to be the *σ* words of *κ*, starting with κ_q, i.e., $\kappa^{(q)} = \kappa_q,...,\kappa_{\sigma+q-1}$. For any $\kappa \in S_{2^{16}}{}^{\sigma+1}$ and $M \in S_{2^{16}}{}^{\sigma}$, $\mathcal{H}^n(\kappa,M)$ is computed by computing $\mathcal{H}^1\left(\kappa^{(1)},M\right),...,\mathcal{H}^1\left(\kappa^{(n)},M\right)$ and concatenating the results. That is:

$$\mathcal{H}^n(\kappa,M) = \mathcal{H}^1\left(\kappa^{(1)},M\right) \circ \mathcal{H}^1\left(\kappa^{(2)},M\right) \circ ... \circ \mathcal{H}^1\left(\kappa^{(n)},M\right).$$

11.7.1.3 MMH[16, σ, 4]

It follows directly from Section 11.7.1.2 that:

$$\mathcal{H}^4(\kappa,M) \equiv MMH[16,\sigma,4](\kappa,M)$$

$$\equiv \mathcal{H}^1\left(\kappa^{(1)},M\right) \circ \mathcal{H}^1\left(\kappa^{(2)},M\right) \circ \mathcal{H}^1\left(\kappa^{(3)},M\right) \circ \mathcal{H}^1\left(\kappa^{(4)},M\right)$$

11.7.1.4 Handling Variable-Size Data

In order to handle data of all possible sizes up to a maximum value, the following rules MUST be followed for computing an MMH function:

- If the data is not a multiple of the word size, pad the data up to a multiple of the word size with octets with the value zero.
- If a key is calculated that is larger than needed for a particular message, truncate the key until it is the correct length.

11.7.2 Definition of MMH-MAC

The MMH-MAC is defined in a manner similar to PacketCable [PKT-SEC]; for a message *M*, keystream κ, and a one-time pad Π in Z_{2^n} , where the number of bits of output is *n*: [5]

$$\text{MMH-MAC}(\kappa, M, \Pi) = \mathcal{H}(\kappa, M) + \Pi$$

where the addition is $\bmod 2^n$. For DOCSIS 4.0 technology, n is 64, and $\mathcal{H}(\kappa, \mathcal{M})$ is $\text{MMH}[16,\sigma,4](\kappa, M)$.

11.7.3 Calculating the DOCSIS MMH-MAC

The following is the algorithm used for creating a DOCSIS MMH-MAC. This algorithm shares the same cryptographic properties as the algorithm used by PacketCable to protect RTP streams [PKT-SEC].

Initial conditions:

1. The Sender and the Receiver share a secret, *S*, that is partitioned into two secrets, S_1 and S_2.
2. The Sender and Receiver share two seeds, Δ_1 and Δ_2, which are defined in Section 11.7.5.
3. The message to be protected is *M*.
4. F(<secret>, <seed>) is a pseudo-random generator whose output depends on a shared secret (<secret>) and a seed (<seed>).
5. The length of the MMH MAC output is 4 words or 8 octets.

Steps to create the DOCSIS MMH-MAC:

1. If M is not a multiple of the word size (2 octets), pad the data up to the next multiple of the word size with octets with the value zero.
2. Calculate a keystream, κ = F(S_1 , Δ_1), of sufficient length to generate the MMH function *H* over (κ, *M*). That is, the length of κ is the length of *M* plus 3 words or 6 octets (the output length of the MMH function, less one word).
3. Calculate the value *A*, which is the output of the MMH function *H* = MMH(κ, M).
4. Concatenate $A \circ S_2$.
5. Calculate the first 8 octets of F($A \circ S_2$, Δ_2).
6. Use the output of step 5 as the one time pad Π in the calculation of the MMH-MAC as defined in Section 11.7.2.
7. The DOCSIS MMH-MAC = A + Π where the addition is mod 2^{64} .

[5] Note, however, that the definition is only similar to the PacketCable definition in the case that the output is one 16-bit word.

The following test vectors were created using the initial conditions and steps described above:

```
[Test Vector #1]

Shared Secret in ASCII = Shared secret #1 94476839
Shared Secret in Hex
53 68 61 72 65 64 20 73 65 63 72 65 74 20 23 31
20 39 34 34 37 36 38 33 39

S1 in Hex
53 61 65 20 65 72 74 23 20 34 37 38 39

S2 in Hex
68 72 64 73 63 65 20 31 39 34 36 33

Message in ASCII = DOCSIS 3.0, fulfilling the need for speed
Message in Hex (padded) =
44 4f 43 53 49 53 20 33 2e 30 2c 20 66 75 6c 66
69 6c 6c 69 6e 67 20 74 68 65 20 6e 65 65 64 20
66 6f 72 20 73 70 65 65 64 00

Key in Hex
8f ea 9f 89 3e d7 f4 4d 9c 45 60 2d a9 7d be 3a
99 10 16 6c bc 1b f6 95 26 ca 04 d1 01 94 7f e1
d9 ca 65 99 fa b5 5a f4 40 6f 81 b4 0d c5 ba 7b

MMH64 function output in Hex
62 37 68 38 ee 5d d0 7c

Pad in Hex
c7 39 8f d3 79 47 0d 04

DOCSIS MMH MAC64 output in Hex
29 70 f8 0c 67 a4 dd 80

[Test Vector #2]

Shared Secret in ASCII = Shared secret #2 07782313
Shared Secret in Hex
53 68 61 72 65 64 20 73 65 63 72 65 74 20 23 32
20 30 37 37 38 32 33 31 33

S1 in Hex
53 61 65 20 65 72 74 23 20 37 38 33 33

S2 in Hex
68 72 64 73 63 65 20 32 30 37 32 31

Message in ASCII = The Magic Words are Squeamish Ossifrages
Message in Hex
54 68 65 20 4d 61 67 69 63 20 57 6f 72 64 73 20
61 72 65 20 53 71 75 65 61 6d 69 73 68 20 4f 73
73 69 66 72 61 67 65 73

Key in Hex
e0 9d 69 6b ec 91 d0 09 3c a7 ed 10 e2 e9 cd f2
6c 23 b8 79 d8 d7 28 b0 b9 8d 3f 6a 18 9d a7 56
b3 55 5a 1a b2 22 68 ff 21 54 fa 99 7b cd

MMH64 function output in Hex
37 de 69 df da db 43 54
```

```
Pad in Hex
4b eb 8e a8 f4 71 4c 32

DOCSIS MMH MAC64 output in Hex
83 c9 f8 88 cf 4c 8f 86
```

11.7.4 MMH Key Derivation for CMTS Extended MIC

The CMTS MUST derive the MMH key for the CMTS Extended MIC as defined in this section. The CMTS and the backend provisioning system share a secret key, $\kappa_{CMTS-EMIC}$. This is equivalent to the shared secret *S* in Section 11.7.3.

From $\kappa_{CMTS-EMIC}$, the CMTS derives two shared secrets, equivalent to S_1 and S_2 in Section 11.7.3:

$$S_1 = S[0] \circ S[2] \circ S[4] \circ \ldots \circ S[n-2]$$

$$S_2 = S[1] \circ S[3] \circ S[5] \circ \ldots \circ S[n-1]$$

where *S*[*i*] is the ith octet of *S* (with respect to 0) and *S* contains *n* octets.

The CMTS uses the ASCII-encoded string "CMTS-EMIC" as the value of Δ_1 in the calculation of the MMH-MAC.

The CMTS uses the ASCII-encoded string "CMTS-EMIC-PAD" as the value of Δ_2 in the calculation of the MMH-MAC.

The CMTS uses the function *F* as defined in Section 11.7.6 as the pseudo-random number generator when calculating the MMH-MAC.

11.7.5 Shared Secret Recommendations

Although this specification emplaces no limitations or requirements on the length or contents of the secret that is shared between the backend provisioning system and the CMTS, it assumes that the secret contains sufficient entropy to ensure adequate cryptographic security when the secret is used in the cryptographic calculations contained in this document. In order to meet this assumption, it is recommended that the shared secret:

1. Be a pseudo-random binary value (as opposed to ASCII or some other simple encoding system that encodes certain bit positions to predictable values); and
2. Have a length of at least 16 octets.

In the event that the output of a hash function over some simple ASCII-encoded message is used as the shared secret (which is *not* recommended), the input message should contain at least 160 characters.

11.7.6 Key Generation Function

Key derivation sections in this document refer to a function F(*S, seed*) where *S* is a shared secret from which keying material is derived, and seed is a constant string. The output of F() is a pseudo-random sequence suitable for use as a key. The output of F(*S, seed*) is generated as follows:

1. From *S*, generate a derived shared secret, *S'*, by accumulation as follows: the value *S'* is obtained by XORing every 16 octets of *S*, padding with zeroes as necessary (i.e., add zeroes to the end of *S* to pad it out to a length that is an integral multiple of 16 octets).
2. Use *S'* as the initial key to the AES-128 (i.e., 128-bit key, 128-bit block) algorithm operating in counter (CTR) mode.
3. For each block of output needed, set the IV equal to the value of the seed, truncated to 128 bits or zero-extended to 128 bits as necessary, and XORed with the number of the block, starting with 1 (one) and incrementing by 1 (one) for each block processed.

The output of F(*S, seed*) is the concatenation of the blocks of output obtained in step 3. Any unused octets at the end of the last repetition of step 3 are discarded.

12 PHYSICAL PROTECTION OF KEYS AND SECURITY PARAMETERS WITHIN THE CM

12.1 Key Storage

CMs MUST store and maintain the CM Device Certificate RSA private/public key pairs.

The CM MUST store the CM Device Certificate private keys in a manner that deters unauthorized disclosure and modification.

A CM SHOULD prevent debugger tools from reading the CM Device Certificate private key in production devices by restricting or blocking physical access to memory containing this key.

The CM MUST meet [FIPS 140-2] security requirements for all instances of private and public permanent key storage.

The CM MUST meet [FIPS 140-2] Security Level 1. [FIPS 140-2] Security Level 1 requires minimal physical protection through the use of production-grade enclosures. The reader should refer to the cited document for the formal requirements; however, below is a summary of those requirements.

Under the [FIPS 140-2] classification of "physical embodiments" of cryptographic modules, external CMs are "multiple-chip stand-alone cryptographic modules." [FIPS 140-2] specifies the following Security level 1 requirements for multiple-chip stand-alone modules:

- The chips are to be of production-grade quality, which includes standard passivation techniques (i.e., a sealing coat over the chip circuitry to protect it against environmental or other physical damage);
- The circuitry within the module is to be implemented as a production grade multiple-chip embodiment (i.e., an IC printed circuit board, a ceramic substrate, etc.);
- The module is to be entirely contained within a metal or hard plastic production-grade enclosure, which may include doors or removable covers.

An internal CM (defined in [DOCSIS CMCIv3.0]) would be classified as a [FIPS 140-2] "multiple-chip embedded cryptographic module." The Security Level 1 requirements for these devices are contained in the first two bullets above.

12.2 Persistent Security Attribute Values

The CM MUST store PSA values in a manner that deters unauthorized disclosure and modification.

CMs SHOULD prevent debugger tools from reading PSA values in production devices by restricting or blocking physical access to memory containing these values.

The CM MUST store PSA values such that unauthorized modifications are detected using either a symmetric keyed hash algorithm which is equivalent to or stronger than HMAC-SHA256 or using an asymmetric digital signature algorithm which is equivalent to or stronger than 2048-bit RSA.

The CM MUST provide integrity protection to all PSA values stored on the device.

13 BPI+ X.509 CERTIFICATE PROFILE AND MANAGEMENT

DOCSIS technology employs [X.509] version 3 digital certificates for authenticating key exchanges between CM and CMTS (see [X.509]). [X.509] is a general-purpose standard; the DOCSIS certificate profile, described here, further specifies the contents of the certificate's defined fields. This certificate profile also defines the hierarchy of trust for the management and validation of DOCSIS certificates.

Except where otherwise noted in Appendix III, DOCSIS certificates comply with [RFC 5280].

The DOCSIS certificate profile draws extensively from the Secure Electronic Transaction (SET) system (see [SET Book 2]). The overall organization of this section, as well as some of the section's contents, reflect that system.

13.1 BPI+ Certificate Management Architecture Overview

The DOCSIS certificate management architecture for BPI+ CM authentication consists of two public key infrastructures (PKIs): a legacy PKI to support backward compatibility with devices implementing older versions of DOCSIS technology and a new PKI that provides stronger cryptography algorithms and key sizes. The legacy PKI is defined by the DOCSIS 3.0 Security specification [DOCSIS SECv3.0]. The new PKI was first defined in [DOCSIS SECv3.1] and is carried forward into this specification. The new PKI consists of a three-level hierarchy of trust supporting three types of certificates:

- Root CA Certificate
- Device CA Certificate
- CM Device Certificates

The Root CA Certificate is used as a trust anchor for the PKI and issues the Device CA Certificate which issues the CM Device Certificates. The new PKI uses a "centralized" model where the Device CA is hosted by CableLabs or an approved 3rd party which issues CM Device Certificates to approved manufacturers. CableLabs manages the new PKI and the certificates issued from its CAs (CableLabs Root CA and CableLabs Device CA, see Appendix III). The legacy PKI is also managed by CableLabs.

The Root CA will also be used as a trust anchor for issuing and validating CA and Code Verification Certificates (CVCs) for the Secure Software Download (SSD) process specified in Section 14.

The Root CA generates and distributes to operators a Certificate Revocation List (CRL), identifying revoked manufacturer certificates. The manner in which CRLs are distributed is outside the scope of this specification. In order to reduce the burden on CM devices that are designed to work in multiple geographic regions, an effort will be made to consolidate the DOCSIS 3.1 PKI hierarchy such that the same BPI+ device certificate for DOCSIS 3.1 will also be valid for EuroDOCSIS 3.1 and other international versions of DOCSIS 3.1 and above.

There are situations where an operator would need CM devices to authenticate to a cloud server outside of BPI+ to enable some services on those devices.

The CM MAY support the use of the CM Device Certificate to authenticate the compliant DOCSIS CM device to a cloud server.

The same requirements for physical protection of the private keys as defined in Section 12 would apply to this use case as well.

The CM MAY support additional CA certificates for the purpose of authenticating the identity of a cloud server.

The authentication mechanism is outside the scope of this specification.

The CM MUST NOT expose the DOCSIS private key to any party when using a non-DOCSIS trust anchor to authenticate its identity to a cloud server.

When the server side is deployed within the operators' network, the use of DOCSIS certificates is allowed. In this configuration, no change is needed to the trust store in the DOCSIS device and applications can rely on the DOCSIS trust store in the device. The CM can use a non-DOCSIS trust store, including a non-DOCSIS Root, to authenticate the identity of servers. The authentication mechanism is outside the scope of this specification. How the CM is securely provisioned with this non-DOCSIS trust store is outside the scope of this specification. For example, the CM might include the additional CA certificates in a software image.

The CM MUST store non-DOCSIS trust anchors in a trust store separate from the DOCSIS trust store.

The CM MUST implement the same level of security for the non-DOCSIS trust store it uses for the DOCSIS trust store.

The use of strong cryptographic algorithm to secure the communication-channel against eavesdropping (i.e., AES-128 or more secure) is required. The use of client-side authentication without proper server authentication is prohibited.

The CM MUST provide either an equivalent or higher level of security to standard protocols widely adopted over either the Internet or in the broadband community.

The CM MUST use TLS v1.2 or newer when authenticating with a cloud server.

The CM MUST use SSH v2 or newer when authenticating with a cloud server.

13.2 Cable Modem Certificate Storage and Management in the CM

The CM MUST have two factory installed CM Device Certificates (and their associated private keys).

The CM MUST have a CM Device Certificate installed that is issued from the new PKI.

The CM MUST have a CM Device Certificate installed that is issued from the legacy PKI.

The CM MUST have the same RSA public key in the CM Device Certificate as the RSA public key in the BPKM Attributes depending upon which CM Device Certificate is used for authentication.

The CM MUST use the CM Device Certificate issued from the new PKI when authenticating with a DOCSIS 3.1 or higher CMTS. The CM is to use the CM Device Certificate issued from the legacy PKI when authenticating with a DOCSIS 3.0 or older version of DOCSIS CMTS.

The CM's non-volatile memory MUST contain a Root CA certificate for SSD image verification.

The CM's non-volatile memory MUST have two CA certificates (the Device CA Certificate from the new PKI and the Manufacturer CA Certificate from the legacy PKI) that signed the CM Device Certificates.

The CM MUST use the Device CA Certificate issued from the new PKI when authenticating with a DOCSIS 3.1 or higher CMTS and use the Manufacture CA Certificate issued from the legacy PKI otherwise. The CM MAY be capable of updating or replacing the Device CA Certificate or the legacy Manufacturer CA Certificate via the DOCSIS code download file (see Section 14, Secure Software Download (SSD)).

The CM MAY embed Device CA or legacy Manufacturer CA certificates.

The CM MUST be able to process certificate serial number values containing 20 octets or fewer.

The CM MUST accept certificates that have serial numbers that are negative or zero.

13.3 Certificate Processing and Management in the CMTS

BPKM employs digital certificates to allow CMTSs to verify the binding between a CM's identity (encoded in a digital certificate's subject name) and its public key. The CMTS does this by validating the CM Device Certificate's certification path. This path will typically consist of three chained certificates: the CM Device Certificate, the Device CA certificate (Mfg CA certificate for legacy PKI), and the Root CA certificate (see Section 13.1). Validating the chain follows the "Basic Path Validation" rules defined in [RFC 5280].

The CMTS MUST support validating certificate chains from the legacy PKI defined in DOCSIS 3.0 and the new PKI defined in this specification.

[RFC 4131] requires that CMTSs support administrative controls that allow the operator to override certification chain validation by identifying a particular CA or CM Device Certificate as trusted or untrusted. This section specifies the management model for the exercise of these controls, as well as the processing a CMTS undertakes to assess a CM Device Certificate's validity, and thus verify the binding between the CM's identity and its public key.

The CMTS MUST be able to process certificate `serialNumber` values containing 20 octets or fewer.

The CMTS MUST accept certificates that have serial numbers that are negative or zero.

With the expiration of DOCSIS Backward Compatible PKI MFG CA certificates beginning in May 2021, some manufacturers have requested and received MFG CA certificates with extended validity periods. These MFG CA certificates with extended validity periods differ from the original MFG CA certificates only in the following attributes:

1. The certificate expiration date
2. The certificate signature
3. The certificate serial number

Since the two MFG CA certificates, the original one and the one with extended validity period, have different attributes they are considered different and unique certificates. The following requirements are to clarify that all unique MFG CA certificates are required to be retained by the CMTS and the CMTS independently applies administrative controls to each unique certificate.

The CMTS MUST retain in the CMTS CA Cert Table all unique MFG CA certificates learned through cable modem authentication messages.

The CMTS MUST independently apply administrative controls to all unique MFG CA certificates in the CMTS CA Cert Table.

Appendix III describes the format of the subject name field for each type of DOCSIS certificate. The issuer field of a certificate matches exactly the subject field of the issuing certificate. New PKI certificates transmitted by a CM in an Auth Info or Auth Request message have name fields that conform to the format described in Appendix III.

A CMTS MUST be capable of processing the name fields of a certificate if the name fields conform to the indicated format in Appendix III.

A CMTS MAY choose to accept a certificate that has name fields that do not conform to the indicated format in Appendix III.

The CMTS MUST process certificate extensions as defined by [RFC 5280] (see section III.1 for certificate profile and extension definitions).

13.3.1 CMTS Certificate Management Model

The CMTS holds copies of the Root CA, Device CA (Mfg CA for legacy PKI) and CM Device Certificates (see Section 13.1), which it obtains through either provisioning or BPKM messaging. Each certificate learned by a CMTS MUST be assigned one of four states: Untrusted, Trusted, Chained, or Root.

The CMTS MUST support the ability to provision at least two Root CA Certificates.

The CMTS MUST support the ability to display the entire Root Certificate(s) and/or its thumbprint to the operator.

A CMTS learns of Device CA certificates through either the CMTS's provisioning interface or through receipt and processing of client CMs' Authentication Information messages.

Regardless of how a CMTS obtains its Device CA certificates, the CMTS MUST mark them as either Untrusted, Trusted or Chained.

If a CA Certificate is not self-signed, the CMTS MUST mark the certificate as Chained.

The CMTS, however, MUST support administrative controls that allow an operator to override the Chained marking and identify a given CA certificate as Trusted or Untrusted.

If a Device CA Certificate is self-signed, the CMTS MUST mark the certificate as either Trusted or Untrusted, according to administratively controlled CMTS policy.

A CMTS obtains copies of CM Device Certificates in the Authorization Requests it receives from CMs. CM Device Certificates are issued by a Device CA. Thus, the CMTS MUST mark CM Device Certificates as Chained unless overridden by CMTS administrative control and configured as Trusted or Untrusted.

13.3.2 Certificate Validation

The CMTS validates the certification paths of CA and CM Device Certificates using Basic Path Validation rules defined in [RFC 5280] and the criteria below.

The CMTS MUST label CA and Cable Modem Certificates as Valid or Invalid if their certification paths are valid or invalid respectively.

Trusted certificates, provisioned in the CMTS, MUST be Valid; this is true even if the current time does not fall within the Trusted certificate's validity period.

Untrusted certificates, provisioned in the CMTS, MUST be Invalid.

The CMTS MUST mark a chained certificate as Valid only if:

1. The certificate chains to a Root CA, Trusted, or Valid certificate that has not been revoked as defined by the Basic Path Validation section in [RFC 5280]; and
2. The current time falls within the validity period of each Chained or Root certificate within the certificate chain; and
3. The certificate is not identified as revoked (see Section 13.4, Certificate Revocation); and
4. In the case of a CM Device Certificate, the CM MAC address encoded in its tbsCertificate.subject field and RSA public key encoded on its tbsCertificate.subjectPublicKeyInfo field match the CM MAC address and RSA public key encoded in the Authorization Request's BPKM Attributes; and
5. In the case of a CM Device Certificate, if the KeyUsage extension is present, the digitalSignature and/or keyAgreement bits are turned on, the keyEncipherment bit is turned on, and the keyCertSign and cRLSign bits are off; in the case of a Device CA Certificate, if the KeyUsage extension is present, the keyCertSign bit is turned on.

The CMTS MUST implement an administrative control option to ignore when the current time falls within the validity period of each Chained or Root certificate within the certificate chain.

If validity period checking is enabled and the time of day has not been acquired by the CMTS, a (non-permanent) authorization reject message MUST be returned by the CMTS in response to an authorization request.

The CMTS MUST NOT invalidate certificates that have non-specified critical extensions (contrary to [RFC 5280]) as long as the certificates satisfy the validity criteria above.

13.4 Certificate Revocation

Providing a mechanism for certificate revocation is a normal part of PKI management. When a certificate is issued, it is expected to be in use for its entire validity period. However, various circumstances may cause a certificate to become invalid prior to the expiration of the validity period. Such circumstances include change of name, change of association between subject and CA, and compromise or suspected compromise of the corresponding private key. Under such circumstances, the CA needs to revoke the certificate. Two methods of supporting certificate revocation are defined in this specification: Certificate Revocation Lists (CRLs) and Online Certificate Status Protocol (OCSP). The CMTS MUST support configuration of none, one, or both certificate revocation methods to be enabled at the same time.

13.4.1 Certificate Revocation Lists

[RFC 5280] defines a method for revoking certificates using [X.509] Certificate Revocation Lists (CRLs).

Figure 13 shows a framework for managing and distributing CRLs. A CRL is a digitally signed, timestamped list of certificate serial numbers revoked by a Certificate Authority (CA). When a CA identifies the compromised certificates, the CA could generate the CRLs itself, or a CA could delegate the CRL generation to a third party CRL Issuer. The CRL Repository is a system that maintains a database of revoked certificates. The interface between the CA or CRL Issuer and the CRL Repository is outside the scope of this specification.

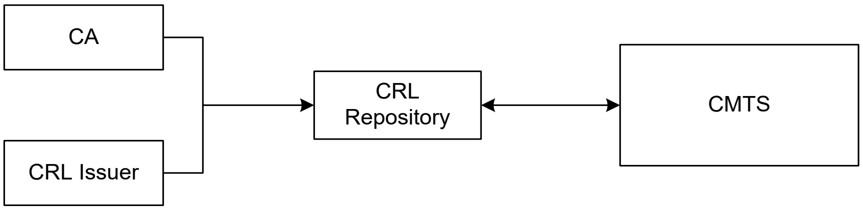

Figure 13 - CRL Framework

The CMTS retrieves CRL entries from the CRL Repository and uses this information to verify if a certificate received during the CM registration process is revoked.

13.4.1.1 CMTS CRL Support

The CMTS MUST support retrieval of CRL files formatted as defined in [RFC 5280]. CRL files may identify revoked certificates that were issued from different CAs.

The CMTS MUST support extensions related to indirect CRL files as defined in [RFC 5280].

The CMTS MUST support HTTP as defined in [RFC 2616] for downloading CRL files.

Before using the information in a CRL file, the CMTS MUST verify that its digital signature chains to a trusted root CA. Trusted root CAs are administratively provisioned in the CMTS.

If the CRL file digital signature cannot be verified, the CMTS MUST discard the CRL file.

The CMTS MUST validate if a CA certificate or CM Device Certificate is revoked during the certificate validation process specified in Section 13.3.2, Certificate Validation.

If the CRL contains the `nextUpdate` value the CMTS MUST refresh the CRL after the specified time has passed.

If the CMTS fails to retrieve the new CRL, the CMTS MUST log an event [DOCSIS CCAP-OSSIv4.0] and continue to use its current CRL. If the CMTS fails to retrieve the new CRL it should attempt to retry retrieval of the CRL file on a periodic basis.

If the CRL does not contain the `nextUpdate` value, the CMTS MUST refresh the CRL according to the configured value as defined in [DOCSIS CCAP-OSSIv4.0].

When the CMTS is configured to use a CRL the CMTS MUST attempt to retrieve the CRL file each time it starts up. During CMTS startup it is possible that some CMs may perform BPI+ authorization before the CRL file has been retrieved.

When the CMTS is configured to use a CRL and a CM's device certificate chain is validated during CMTS startup before the CRL file is retrieved the CMTS MUST log an event for that CM [DOCSIS CCAP-OSSIv4.0] and bypass CRL checking.

13.4.2 Online Certificate Status Protocol

[RFC 6960] defines an Online Certificate Status Protocol (OCSP) for querying the status of a digital certificate. The CMTS sends a certificate status request to an OCSP responder when it receives a CA certificate or a CM Device Certificate (see Figure 14). The OCSP responder sends a status response indicating that the certificate is "good," "revoked," or "unknown." The OCSP responder checks only the revocation status of a certificate; it does not verify the validity of the certificate itself. The CMTS uses the result from the OCSP responder during certificate validation process specified in Section 13.3.2.

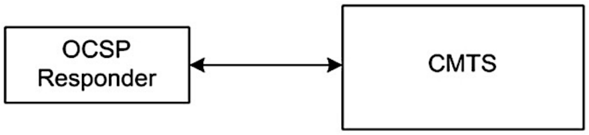

Figure 14 - OCSP Framework

The CMTS MUST be capable of acting as an OCSP client as defined in [RFC 6960].

The CMTS SHOULD cache the OCSP response status for a certificate if the `nextUpdate` value is present in the OCSP response.

If the CMTS caches the OCSP response status for a given certificate, the CMTS MUST retrieve the revocation status from the cache.

Once the `nextUpdate` time for that certificate has passed, the CMTS MUST continue using the revocation status value from the cache until an update is retrieved from the OCSP Responder.

If the CMTS is unable to retrieve the OCSP status for an uncached certificate or if the retrieved status is "unknown" the CMTS MUST log an event (see [DOCSIS CCAP-OSSIv4.0]) and assume the certificate status to be "good".

If the `nextUpdate` value is not present in the OCSP response, the CMTS MUST NOT cache the OCSP response status for a certificate.

If the CMTS is configured with OCSP Responder information, the CMTS MUST send an OCSP request when a CA certificate or CM Device Certificate is obtained using the Authentication Information message, or Authentication Request message, respectively, unless there is a valid certificate status in the cache.

When the CMTS is attempting to communicate with the OCSP Responder, the exchange is not to significantly delay the CM provisioning process.

If no OCSP response is received, the CMTS MUST proceed using the currently cached revocation status.

For uncached certificate states, the CMTS MUST proceed as if an OCSP response with the status "good" has been received.

The CMTS MUST support OCSP over HTTP as described in [RFC 6960].

The CMTS MAY generate a signature in the OCSP request.

The CMTS MUST bypass validation of the signature in an OCSP response based on the configured value as defined in [DOCSIS CCAP-OSSIv4.0].

14 SECURE SOFTWARE DOWNLOAD (SSD)

14.1 Introduction

DOCSIS technology supports downloading code to CMs. Authenticating the source and verifying the integrity of downloaded code is vital to the overall operation and security of DOCSIS-based networks. Code is signed with a certificate from the new PKI (see Section 13.1) and then validated by the CM.

The software download module is an attractive target for an attacker. If an attacker were able to mount a scalable attack against the software download module, he could potentially install code to disable all the CMs within a domain or disrupt service on a wide scale. To thwart these attacks, the attacker is forced to overcome several security barriers.

14.2 Overview

The requirements defined in this section address these primary security goals for the code download process:

- The CM needs to have a means to authenticate that the originator of any download code is a known and trusted source;
- The CM needs to have a means to verify that the downloaded code has not been altered from the original form in which it was provided by the trusted source;
- The process needs to strive to simplify the operator's code file-handling requirements and provide mechanisms for the operator to upgrade or downgrade the code version of cable modems on their network;
- The process allows operators to dictate and control their policies with respect to: (a) which code files will be accepted by CMs within their network, and (b) security controls that define the security of the process on their network;
- CMs are able to move freely among systems controlled by different operators;
- Support updating the Root CA Certificate in the CM (optional);
- Support updating the Device CA Certificate in the CM (optional);
- Support updating the legacy Root CA Public Key in the CM (optional);
- Support updating the legacy Manufacturer CA Certificate in the CM (optional).

This document limits its scope to these primary system security requirements but acknowledges that in some cases additional security may be desired. The concerns of individual operators or CM manufacturers may result in additional security related to the distribution or installation of code into a CM or other DOCSIS network element. This specification does not restrict the use of further protections, as long as they do not conflict with the requirements of this specification.

Multiple levels of protection are required to protect and verify the code download:

- The manufacturer of the CM code always applies a digital signature to the code file. The signature is verified with a certificate chain that extends up to the Root CA. The manufacturer signature affirms the source and integrity of the code file to the CM;
- Though the manufacturer always signs its code file, an operator may later apply its code signature in addition to the manufacturer signature. If a second signature is present, the CM verifies both signatures with a certificate chain that extends up to the Root CA before accepting a code file;
- OSS mechanisms for the provisioning and control of the CM are important to the proper execution of this process. The code-upgrade capability of a CM is enabled during the provisioning and registration process. Code downloads are initiated during the provisioning and registration process or can be initiated in normal operation using an SNMP command.

The code file is built using a PKCS#7-compliant structure that is defined below and in [PKCS#7]. This data structure is referred to as the Firmware Authentication Header (FWAH). Included in this structure are:

- The code image; i.e., the upgrade code image;
- The Code Verification Signature (CVS); i.e., the digital signature over the code image and any other authenticated attributes as defined in the structure;
- The Code Verification Certificate (CVC); i.e., an [X.509]-compliant certificate that is used to deliver and validate the public code verification key that will verify the signature over the code image. The DOCSIS Certificate Authority, a trusted party whose public key is already stored in the CM, signs this certificate.

Figure 15 shows the basic steps required for the signing of a code image when the code file is signed only by the CM manufacturer, and when the code file is signed by the CM manufacturer and co-signed by an operator.

With DOCSIS technology, the Root CA certificate is installed in each CM as a trust anchor. The code manufacturer builds the code file by signing the code image using a DOCSIS PKCS#7 digital signature structure with a Mfr CVC certificate and the issuing CVC CA certificate. The code file is then sent to the operator.

The operator verifies that the code file is from a trusted DOCSIS manufacturer and has not been modified. At this point, the operator has the option of loading the code file on the software download server as-is, or of adding its signature and operator CVC and issuing CVC CA certificate to the code file. During the code upgrade process, the CM will retrieve the code file from the software download server and verify the new code image using the Root CA Certificate trust anchor before installing it. See Appendix III for CVC chain details.

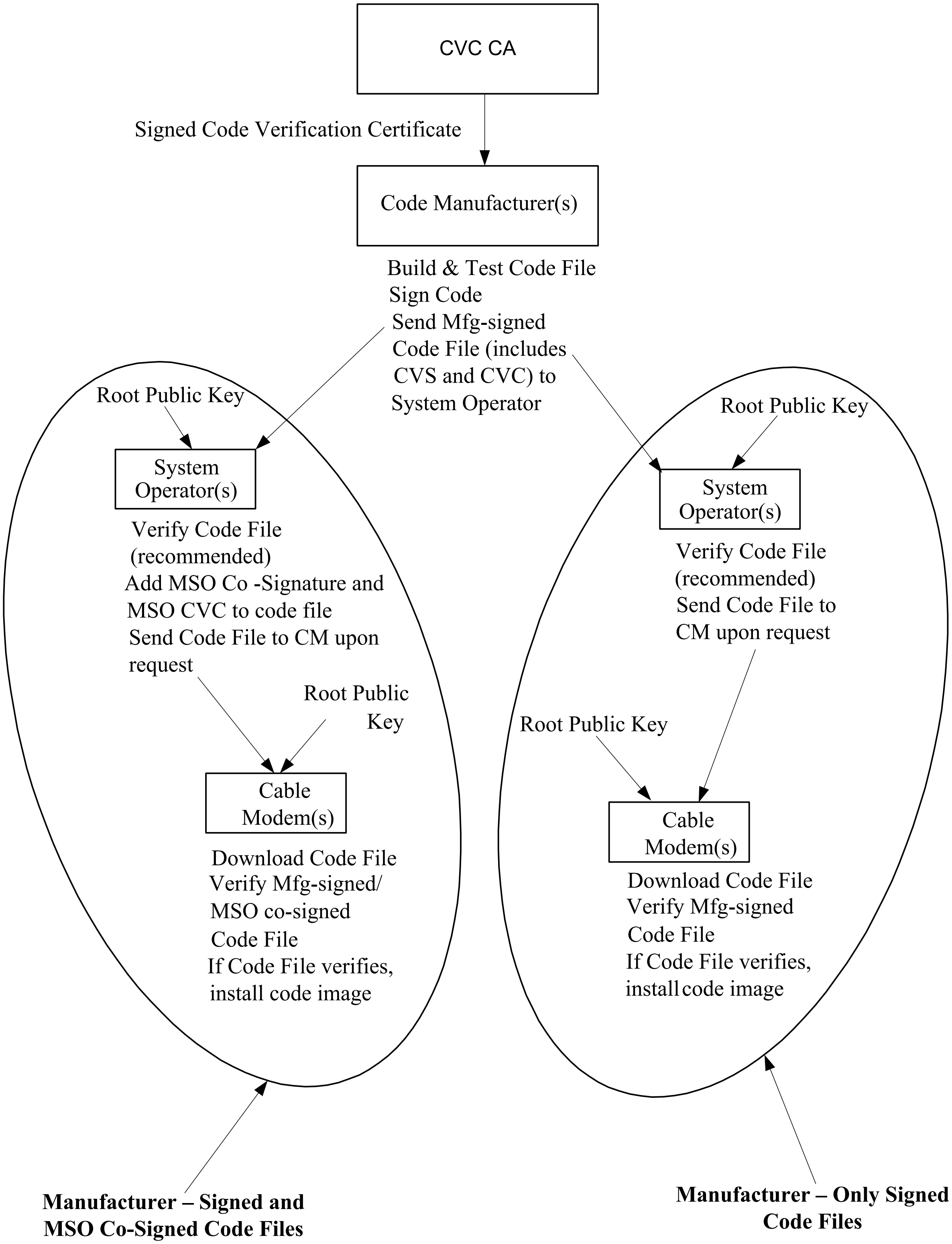

Figure 15 - Typical Code Validation Hierarchy

14.3 Software Code Upgrade Requirements

The following sections define requirements of the CM software code upgrade verification process. All code upgrades are prepared and verified as described. The new PKI used for issuing CVCs consists of three types of certificates: a Root CA, CVC CA, and the CVC. CableLabs manages the new PKI and the certificates issued from its CAs (CableLabs Root CA and CableLabs CVC CA; see Appendix III for certificate profile and extension definitions). Note: the CableLabs Root CA is used to issue both CM Device Certificates and CVC certificates.

A CM MUST verify code upgrades according to this specification regardless of whether Baseline Privacy is enabled or disabled.

The CM MUST process CVC extensions as defined by [RFC 5280].

14.3.1 Code File Processing Requirements

The code file format is defined in Appendix III.

The CM MUST reject the DOCSIS code file if the `signedData` field does not match the DER encoded structure represented in Appendix III.

The CM MUST be able to verify DOCSIS code file signatures that are signed using key modulus lengths of 2048, 3072, and 4096 bits. The public exponent is F_4 (65537 decimal).

The CM MUST reject the CVC if it does not match the DER encoded structure represented in Appendix III.

The CM MUST NOT install the upgraded code image unless the code image has been verified as being compatible with the CM.

If the code download and installation is successful, then the CM MUST replace its currently stored Root CA Certificate with the Root CA Certificate in the `SignedContent` field, if one was present.

If the code download and installation is successful, then the CM MUST replace its currently stored Device CA Certificate with the Device CA Certificate received in the `SignedContent` field, if any were present.

If the code download and installation is successful, then the CM MUST replace its currently stored legacy Manufacturer CA certificate with the legacy Manufacturer CA certificate received in the `SignedContent` field, if any were present.

14.3.2 Code File Access Controls

In addition to the cryptographic controls provided by the digital signature and the certificate, special control values are included in the code file for the CM to check before it accepts a code image as valid.

The conditions placed on the values of these control parameters MUST be satisfied before the CM attempts to validate the CVC and the CVS (see Sections 14.3.2.1, Subject Organization Names and 14.3.2.2, Time Varying Controls).

Some of these special control value conditions are maintained separately for the legacy PKI and the new PKI (see Section 13).

14.3.2.1 Subject Organization Names

The CM MUST recognize up to two names that it considers a trusted code-signing agent if present in the subject field of a code file CVC.

These are:

The cable modem manufacturer:

A manufacturer CVC is always included in the code file.

The CM MUST verify that the manufacturer name in the manufacturerCVC subject field exactly matches the manufacturer name stored in the CM's non-volatile memory by the manufacturer.

The CM MUST store separate manufacturer names, one for the legacy PKI and one for the new PKI, in non-volatile memory.

A co-signing agent:

As described above, DOCSIS technology permits another trusted organization to co-sign code files destined for the CM. In most cases this organization is the operator. The organization name of the co-signing agent is communicated to the CM via a co-signer CVC in the configuration file when initializing the CM's code verification process.

The CM MUST verify that the co-signer organization name in the co-signer CVC subject field, exactly matches the co-signer organization name previously received in the co-signer initialization CVC, and stored by the CM.

14.3.2.2 Time Varying Controls

In support of the code upgrade process, the CM MUST keep two UTC time values associated with each code-signing agent. These values are known as `codeAccessStart` and `cvcAccessStart`.

The CM MUST store and maintain one pair of time values for the CM manufacturer signing agent.

If the CM is assigned a code co-signing agent, the CM MUST maintain a pair of time values for the code co-signing agent.

These values are used to control code file access to the cable modem by individually controlling the validity of the CVS and the CVC.

Stored and maintained time values in the CM MUST have a precision of one second.

Stored and maintained time values in the CM MUST be capable of representing all times (with one second precision) between midnight, Jan 1 1950, and midnight Jan 1 2050.

The CM MUST NOT allow the values of `codeAccessStart` and `cvcAccessStart` corresponding to the cable modem's manufacturer signing agent to decrease.

The CM MUST NOT allow the value of `codeAccessStart` and `cvcAccessStart` corresponding to the co-signing agent to decrease as long as the co-signing agent does not change and the CM maintains the co-signer time-varying control values (see Section 14.3.3, Cable Modem Code Upgrade Initialization).

For the manufacturer code signing agent, the CM MUST store and maintain time-varying control values for the new PKI in non-volatile memory.

14.3.3 Cable Modem Code Upgrade Initialization

Before the cable modem can upgrade code, it should be properly initialized. Its manufacturer first initializes the cable modem.

Every time a CM registers on a DOCSIS network, the CM MUST check its current initialization state with respect to the operational needs of the particular network.

It may be necessary for the cable modem to reinitialize at registration, particularly if the cable modem has moved from one network to another.

14.3.3.1 Manufacturer Initialization

It is the responsibility of the manufacturer to install the initial code version in the CM.

In support of code upgrade verification, the CM MUST have values for the following parameters loaded into the CM's non-volatile memory:

- CM manufacturer `organizationName`;
- `codeAccessStart` initialization value;
- `cvcAccessStart` initialization value.

The CM MUST initialize the values of `codeAccessStart` and `cvcAccessStart` for the new PKI to a UTCTime equal to the validity start time of the manufacturer's latest CVC.

These values will be updated periodically under normal operation via manufacturer CVCs that are received and verified by the cable modem.

14.3.3.2 Network Initialization

The method for obtaining CM code download files is defined in [DOCSIS MULPIv4.0]. The CM receives settings relevant to code upgrade verification in its configuration file.

The CM MUST NOT use these settings until after the CMTS has successfully registered the CM.

The configuration file normally includes the most up-to-date CVC applicable for the destination cable modem. When the configuration file is used to initiate a code upgrade, it will include a CVC to initialize the cable modem for accepting code files according to this specification.

Alternatively, the configuration file can directly include the FWAH of the firmware upgrade (see Table 72) for the destination cable modem. In this case, the signing CVC and its issuing CVC CA certificates (together with the co-signer chain, if co-signed) are directly included in the FWAH.

Regardless of whether a code upgrade is required, a CVC in the configuration file MUST be processed by the cable modem.

A configuration file can contain:

- No CVCs, or
- The Manufacturer CVC Chain and/or the Co-signer CVC Chain from the new PKI (Type 81 and 82), or
- The FWAH of a firmware update file that already contains the CVC chains (Type TBD_B).

In addition to the authentication information, the configuration file can also contain:

- (Optional) Revocation Information (OCSP-CVC-Responses) for the CVC chains (Type TBD_A)

Before the CM will enable its ability to upgrade code files from the network, a CM MUST receive either a valid CVC or a valid FWAH in a configuration file and successfully register with the CMTS.

If the CM's configuration file does not contain either a valid CVC or a valid FWAH and its ability to upgrade code files has been disabled, then the CM MUST reject any information in either a CVC or a FWAH delivered via SNMP.

When the cable modem's configuration file does not contain either a co-signer CVC or a FWAH that carries a co-signer signature, the CM MUST NOT accept code files that have been co-signed.

If the CM is configured to accept code co-signed by a code-signing agent, the following parameters MUST be stored in the CM's memory when the co-signer CVC is processed:

- Co-signing agent's `organizationName`;
- Co-signer `cvcAccessStart`;
- Co-signer `codeAccessStart`.

Unlike the manufacturer organizationName and time varying control values, the co-signer organizationName and time varying control values are not required to be stored in non-volatile memory.

14.3.3.2.1 Processing the Configuration File CVC

When a CVC is included in the configuration file, the CM MUST verify the CVC before accepting any of the code upgrade settings it contains.

Upon receipt of the CVC the CM MUST perform the following validation and procedural steps.

If any of the following verification checks fail, the CM MUST immediately halt the CVC verification process.

If the CM configuration file does not include either a valid CVC or a valid FWAH, the CM MUST NOT download upgrade code files, whether triggered by the CM configuration file or via an SNMP MIB.

If the CM configuration file does not include either a valid CVC or a valid FWAH, the CM SHOULD NOT process CVCs or FWAH subsequently delivered via an SNMP MIB.

If the CM configuration file does not include either a valid CVC or a valid FWAH, the CM MUST NOT accept information from a CVC or FWAH subsequently delivered via an SNMP MIB.

Following receipt of a CVC in a configuration file, and after the CM has successfully registered with the CMTS, the CM MUST:

1. Verify that the Extended Key Usage extension is present in the CVC, as specified in Appendix III.
2. Verify that the manufacturer CVC validity start time is greater than or equal to the manufacturer `cvcAccessStart` value currently held in the CM if the CVC is a Manufacturer CVC and the subject `organizationName` is identical to the CM's manufacturer name.
3. Reject this CVC and log an error if the CVC is a Manufacturer CVC and the subject `organizationName` is not identical to the cable modem's manufacturer name.
4. Verify that the validity start time is greater than or equal to the co-signer `cvcAccessStart` value currently held in the CM if the CVC is a co -signer CVC and the subject `organizationName` is identical to the CM's current code co-signing agent.
5. After the CVC has been validated (and registration is complete), make this subject organization name become the CM's new code co-signing agent if the CVC is a co -signer CVC and the subject `organizationName` is not identical to the current code co-signing agent name.
6. Verify that both the CVC and the CVC CA Certificate signatures chain up to the Root CA Certificate of the new PKI or the Root CA key of the legacy PKI held by the CM.
7. If time-of-day has been acquired, verify that the validity periods for the CVC certificate, the issuing CVC CA certificate, and the Root CA have not expired.
8. Update the CM's current value of `cvcAccessStart` corresponding to the CVC's subject `organizationName` (i.e., manufacturer or code co-signing agent) with the validity start time value from the validated CVC. If the validity start time value is greater than the CM's current value of `codeAccessStart`, update the CM's `codeAccessStart` value with the validity start time value.

14.3.3.2.2 Processing the SNMP CVC

When the CM is not enabled to upgrade code files it MUST reject all CVCs received via SNMP. CVCs received via SNMP will also chain up to the same Root CA certificate or public key that was used to validate the CVC in the configuration file (see Section 14.3.5).

The CM MUST process CVCs received via SNMP when it is enabled to upgrade code files.

When the CM is not enabled to upgrade code files, the CM MUST reject all CVCs received via SNMP.

When validating a CVC received via SNMP, the CM MUST perform the following validation and procedural steps.

If any of the following verification checks fail, the CM MUST immediately halt the CVC verification process, log the error if applicable, and remove all remnants of the process up to that step.

When a CM receives a CVC via SNMP, the CM MUST:

1. Verify that the Extended Key Usage extension is in the CVC as specified in Appendix III.
2. Verify that the manufacturer CVC validity start time is greater than the manufacturer cvcAccessStart value currently held in the CM if the CVC subject `organizationName` is identical to the CM's manufacturer name.
3. Verify that the validity start time is greater than the co-signer `cvcAccessStart` value currently held in the CM if the CVC subject `organizationName` is identical to the cable modem's current code co-signing agent.

4. Reject this CVC if the CVC subject `organizationName` is not identical to CM's manufacturer or current code co-signing agent name.
5. Verify that the CVC and the CVC CA Certificate signatures chain up to the Root CA Certificate of the new PKI.
6. If the time-of-day has been acquired, verify that the validity periods for the CVC certificate, the issuing CVC CA certificate, and the Root CA certificate have not expired.
7. Update the current value of the subject's `cvcAccessStart` values with the validated CVC's validity start time value.

If the validity start time value is greater than the CM's current value of `codeAccessStart`, the CM MUST replace its `codeAccessStart` value with the validity start value.

14.3.3.2.3 Processing the Configuration File Firmware Auth Header (FAH)

When a Firmware Authentication Header (FWAH) is included in the configuration file via the CodeFile-AuthHeader or Co-AH TLV (Type TBD_A) as described in [DOCSIS MULPIv4.0], the CM MUST verify the contents of the TLV before accepting any of the code upgrade settings it contains according to the following validation and procedural steps.

If any of the verification checks fail, or if any section of the code file is rejected due to invalid formatting, the CM MUST immediately halt the download process and log the error if applicable, remove all remnants of the process to that step, and continue to operate with its existing code.

Following receipt of a FWAH in a configuration file, after the CM has successfully registered with the CMTS, and before downloading the code file, the CM MUST validate the signing certificates (i.e., Manufacturer's and, if present, Co-Signer's) and their certificate chains by following the steps in Section 14.3.3.2.1, Processing the Configuration File CVC.

Following receipt of a FWAH in a configuration file, after the CM has successfully registered with the CMTS, and before downloading the code file, the CM MUST validate the FWAH according to the procedures specified in Section 14.3.5.1, Cable Modem Code Verification Steps with the exception of signature validation in Step 5 and Step 8-e.

When validating the FWAH signature, the CM MUST validate the correctness of the signature but delay matching the signed value (i.e., the encrypted hash calculated over the signed data) with the `SignedContent` (see Appendix III.11.3.2, Signed Content) until the CM has downloaded the code file and the `SignedContent` is available on the CM.

If no error conditions have been detected during FWAH signature validation, the CM can proceed with the download of the code file.

After the code file download is completed, the CM MUST verify that the signed value in the FWAH matches the value calculated over the `SignedContent` of the downloaded code file.

The CM MUST verify that the FWAH provided via the configuration file matches the FWAH at the beginning of the downloaded code file.

In case of FWAH signature validation errors, the CM MUST immediately halt the upgrade process and log the error if applicable, remove all remnants of the process to that step, and continue to operate with its existing code.

After validating the correctness and integrity of the downloaded code file, if no errors have been detected, the CM MUST proceed with the code installation as specified in Section 14.3.5.2, Cable Modem Code File Installation Steps.

14.3.3.2.4 Processing SNMP-delivered Firmware Authentication Header (FWAH)

When the CM is enabled to upgrade code files (e.g., via the use of a CVC certificate in the config file), the CM MUST process the FWAH received via SNMP before downloading the code file.

If any of the verification checks fail, or if any section of the code file is rejected due to invalid formatting, the CM MUST immediately halt the download process and log the error if applicable, remove all remnants of the process to that step, and continue to operate with its existing code.

The CM MUST process the FWAH received via SNMP when it is enabled to upgrade code files.

When the CM is not enabled to upgrade code files, the CM MUST ignore all FWAH received via SNMP.

Following receipt of a FWAH via SNMP, after the CM has successfully registered with the CMTS, and before downloading the code image file, the CM MUST validate the signing certificates (i.e., Manufacturer's and, if present, Co-Signer's) and their certificate chains by following the steps in Section 14.3.3.2.2, Processing the SNMP CVC.

Following receipt of a FWAH via SNMP, after the CM has successfully registered with the CMTS, and before downloading the code file, the CM MUST validate the FWAH according to the procedures specified in Section 14.3.5.1, Cable Modem Code Verification Steps with the exception of signature validation in Step 5 and Step 8-e.

When validating the FWAH signature in Section 14.3.5.1, Cable Modem Code Verification Steps in Step 5 and Step 8-e, the CM MUST validate the correctness of the signature only and delay matching the signed value (i.e., the encrypted hash calculated over the signed data) with the calculated value over the `SignedContent` until the CM has downloaded the code file and the `SignedContent` is available on the CM.

If no error conditions have been detected during validation, the CM can proceed with the download of the code file.

After the code file download is completed, the CM MUST verify that the signed value in the FWAH matches the value calculated over the `SignedContent` of the downloaded code file.

The CM MUST verify that the FWAH provided via SNMP matches the FWAH at the beginning of the downloaded upgrade file.

In case of FWAH validation errors, the CM MUST immediately halt the upgrade process and log the error if applicable, remove all remnants of the process to that step, and continue to operate with its existing code.

After validating the correctness and integrity of the downloaded image, if no errors have been detected, the CM MUST proceed with the code installation as specified in Section 14.3.5.2, Cable Modem Code File Installation Steps.

14.3.4 Code Signing Guidelines

Manufacturer and operator code signing guidelines are provided in Appendix III.

14.3.5 Code Verification Requirements

The CM MUST NOT install upgrade code unless the code has been verified.

14.3.5.1 Cable Modem Code Verification Steps

When downloading code, the CM MUST perform the verification checks presented in this section.

If any of the verification checks fail, or if any section of the code file is rejected due to invalid formatting, the CM MUST immediately halt the download process and log the error if applicable, remove all remnants of the process to that step, and continue to operate with its existing code.

The verification checks can be performed in any order.

1. The CM MUST verify that:
    a) The value of `signingTime` is equal to or greater than the manufacturer `codeAccessStart` value currently held in the CM;
    b) The value of `signingTime` is equal to or greater than the manufacturer CVC validity start time;
    c) The value of `signingTime` is less than or equal to the manufacturer CVC validity end time.
2. The CM MUST verify that:

   a) The manufacturer CVC subject `organizationName` is identical to the manufacturer name currently stored in the CM's memory;

   b) The manufacturer CVC validity start time is equal to or greater than the manufacturer `cvcAccessStart` value currently held in the CM;

   c) The Extended Key Usage extension in the Manufacturer CVC meets the requirements of Appendix III.

3. The CM MUST verify that the Mfr CVC chains up to Root CA held by the CM.

4. If the time-of-day has been acquired, the CM MUST verify that the validity periods for the CVC, the issuing CA certificate, and the Root CA certificate have not expired.

5. The CM MUST verify the manufacturer code file signature. If the signature does not verify, the CM MUST reject all components of the code file (including the code image), and any values derived from the verification process should be immediately discarded.

6. In case the OCSP response TLV is provided in the configuration file or via SNMP SET, the CM MUST validate the revocation status of the certificate(s) in the chain.

7. If time-of-day has not been acquired, the CM MAY ignore the validity periods of the OCSP CVC responses during validation.

8. If the manufacturer signature verifies and a co-signing agent signature is required:

   a) The CM MUST verify that:

      (1) The co-signer signature information is included in the code file;

      (2) The value of `signingTime` is equal to or greater than the corresponding codeAccessStart value currently held in the CM;

      (3) The value of `signingTime` is equal to or greater than the corresponding CVC validity start time;

      (4) The value of `signingTime` is less than or equal to the corresponding CVC validity end time.

   b) The CM MUST verify that:

      (1) The co-signer CVC subject `organizationName` is identical to the co-signer organization name currently stored in the CM's memory;

      (2) The co-signer CVC validity start time is equal to or greater than the `cvcAccessStart` value currently held in the CM for the corresponding subject `organizationName`;

      (3) The Extended Key Usage extension in the co -signer CVC meets the requirements of Appendix III.

   c) The CM MUST verify that the Co-Signing CVC Certificate chains up to the Root CA of the new CA.

   d) If time-of-day has been acquired, verify that the validity periods for the CVC, the issuing CVC CA certificate, and the Root CA certificate have not expired.

   e) The CM MUST verify the co-signer code file signature.

   f) In case the OCSP responses TLV is provided in the configuration file or via SNMP SET, the CM MUST validate the revocation status of the certificate(s) in the chain.

   g) If time-of-day has not been acquired, the CM MAY ignore the validity periods of the OCSP CVC responses during validation.

If no errors have been detected during the validation process, the CM can proceed with the installation of the code file as described in Section 14.3.5.2, Cable Modem Code File Installation Steps.

If the signature does not verify, the CM MUST reject all components of the code file (including the code image), and any values derived from the verification process should be immediately discarded.

7. Once the manufacturer, and optionally the co-signer, signature has been verified, the code image can be trusted and installation may proceed. Before installing the code image, the CM SHOULD immediately discard all other

components of the code file and any values derived from the verification process except the [PKCS#7] signingTime values and the CVC validity start values.

8. The CM can upgrade its software by installing the code file according to [DOCSIS MULPIv4.0].
9. If the code installation is unsuccessful, the CM MUST discard the [PKCS#7] signingTime values and CVC validity start values it just received in the code file. The procedure for handling this failure condition is specified in [DOCSIS MULPIv4.0].
10. Once the code installation is successful, the CM MUST:
    a) Update the current value of manufacturer codeAccessStart with the [PKCS#7] signingTime value;
    b) Update the current value of manufacturer cvcAccessStart with the CVC validity start value.
11. If the code installation is successful, and if the code file was co-signed, the CM MUST:
    a) Update the current value of the co-signer codeAccessStart with the [PKCS#7] signingTime value;
    b) Update the current value of the co-signer cvcAccessStart with the CVC validity start value.

14.3.5.2 Cable Modem Code File Installation Steps

Once the manufacturer, and optionally the co-signer, signature has been verified, the code image can be trusted, and installation can proceed. In order to proceed with the installation of the validated software, the following steps are performed:

1. Before installing the code file, the CM SHOULD immediately discard all other components of the code file and any values derived from the verification process except the FWAH signingTime values and the CVC validity start values.

2. The CM can upgrade its software by installing the code file according to [DOCSIS MULPIv4.0].

3. If the code installation is unsuccessful, the CM MUST discard the FWAH signingTime values and CVC validity start values it just received in the code file. The procedure for handling this failure condition is specified in [DOCSIS MULPIv4.0].

4. Once the code installation is successful, the CM MUST both update the current value of manufacturer codeAccessStart with the FWAH signingTime value; and update the current value of manufacturer cvcAccessStart with the CVC validity start value.

5. If the code installation is successful, and if the code file was co-signed, the CM MUST both update the current value of the co-signer codeAccessStart with the FWAH signingTime value; and update the current value of the co-signer cvcAccessStart with the CVC validity start value.

14.3.6 DOCSIS Interoperability

Images for DOCSIS 4.0 secure software download (SSD) are to be signed using certificates from the new PKI defined in this specification. The CM supports secure software downloads using certificates from only the new PKI.

If the configuration file contains a Manufacturer CVC Chain and/or a Co-signer CVC Chain, the CM MUST perform secure software download.

14.3.7 Error Codes

The CM MUST log the following error events when they occur during the code verification process.

DOCSIS event logging requirements and event message formats are defined in [DOCSIS CM-OSSIv4.0]:

1. Improper code file controls

   Conditions:

   a) CVC subject organizationName for manufacturer does not match the CM's manufacturer name.

b) CVC subject `organizationName` for code co-signing agent does not match the CM's current code co-signing agent.

c) The manufacturer FWAH `signingTime` value is less-than the `codeAccessStart` value currently held in the CM.

d) The manufacturer FWAH validity start time value is less-than the `cvcAccessStart` value currently held in the CM.

e) The manufacturer CVC validity start time is less-than the `cvcAccessStart` value currently held in the CM.

f) The manufacturer FWAH `signingTime` value is less-than the CVC validity start time.

g) Missing or improper extended key-usage extension in the manufacturer CVC.

h) The co-signer FWAH `signingTime` value is less-than the `codeAccessStart` value currently held in the CM.

i) The co-signer FWAH validity start time value is less-than the `cvcAccessStart` value currently held in the CM.

j) The co-signer CVC validity start time is less-than the `cvcAccessStart` value currently held in the CM.

k) The co-signer FWAH `signingTime` value is less-than the CVC validity start time.

l) Missing or improper extended key-usage extension in the co-signer CVC.

2. Code file manufacturer CVC validation failure

   Conditions:

   a) The manufacturer CVC in the code file does not chain to the new PKI.

   b) Either the manufacturer CVC certificate or the CVC CA certificate in the code file has been revoked.

3. Code file manufacturer CVS validation failure

4. Code file co-signer CVC validation failure

   a) The co-signer CVC in the code file does not chain to the new PKI.

   b) The manufacturer CVC in the code file has been revoked.

5. Code file co-signer CVS validation failure

6. Improper configuration file CVC format

   a) Missing or improper key usage attribute.

7. Configuration file CVC validation failure

8. Improper SNMP CVC format

   Conditions:

   a) CVC subject organizationName for manufacturer does not match the CM's manufacturer name,

   b) CVC subject organizationName for code co-signing agent does not match the CM's current code co-signing agent,

   c) The CVC validity start time is less-than or equal-to the corresponding subject's cvcAccessStart value currently held in the CM.

   d) Missing or improper key usage attribute.

9. SNMP CVC validation failure

   Conditions:

   a) The manufacturer CVC received via SNMP does not chain to the same Root CA certificate of the new PKI.

b) The co-signer CVC received via SNMP does not chain to the same Root CA certificate of the new PKI.

14.4 Security Considerations (Informative)

The method(s) used to protect private keys are a critical factor in maintaining security. Users authorized to sign code, i.e., manufacturers and operators who have been issued code verification certificates (CVCs) by the DOCSIS root CA, should protect their private keys. An attacker with access to the private key of an authorized code-signing user can create, at will, code files that are potentially acceptable to a large number of CMs.

The defense against such an attack is for the operator to revoke the certificate whose associated code-signing private key has been learned by the attacker. To revoke a certificate, the operator delivers to each affected CM, an updated CVC with a validity start time that is newer than that of the certificate(s) being revoked. The new CVC can be delivered via any of the supported mechanisms: configuration file, code file, or SNMP. The new CVC implicitly revokes all certificates whose validity start time is earlier than that of the new CVC.

To reduce the vulnerability to this type of attack, operators should regularly update the CVC in each CM, at a frequency comparable to how often the operator would update a CRL if one were available. Regular updates help manage the time interval during which a compromised code-signing key is useful to an attacker. CVCs should also be updated if it is suspected that a code-signing key has been compromised. To update the CVC, the user needs a CVC whose validity start time is newer than the CVC in the CM. This implies that the DOCSIS root CA regularly issues new CVCs to all authorized code-signing manufacturers and operators, to make the CVCs available for update.

When a CM is attempting to register on the network for the first time or after being off-line for an extended period, it should receive a trusted CVC as soon as possible. This provides the CM with the opportunity to receive the most up-to-date CVC available and deny access to CVCs that needed to be revoked since the CM last initialization. The first opportunity for the CM to receive a trusted CVC is in its configuration file. If the configuration file does not include a valid CVC, the CM will not request or have the ability to remotely upgrade code files. In addition, the CM will not accept CVCs subsequently delivered via SNMP.

To mitigate the possibility of a CM receiving a previous code file via a replay attack, the code files include a signing-time value in the Firmware Authentication Header (FWAH) structure as described in [PKCS#7] that can be used to indicate the time the code image was signed. When the CM receives a code file signing-time that is later than the signing-time it last received, it will update its internal memory with this value. The CM will not accept code files with an earlier signing-time than this internally stored value. To upgrade a CM with a new code file without denying access to past code files, the signer may choose not to update the signing-time. In this manner, multiple code files with the same code signing-time allow an operator to freely downgrade a CM's code image to a past version (that is, until the CVC is updated). This has a number of advantages for the operator, but these advantages should be weighed against the possibilities of a code file replay attack.

Without a reliable mechanism to revert back to a known good version of code, any code-update scheme, including the one in this specification, has the weakness that a single, successful forced update of an invalid code image may render the CM useless, or may cause the CM to behave in a manner harmful to the network. Such a CM may not be repairable via a remote code update, since the invalid code image may not support the update scheme.

15 ADDITIONAL REQUIREMENTS

This section defines the Security requirements for DOCSIS 4.0 functionality.

15.1 Increased Security of Keys

15.1.1 Introduction

This service will strengthen the storage and handling of keys in the CM by defining new methods.

15.1.2 Encryption

The CM MUST store any persistently stored copy of the CM device certificate private key using a device-unique key (DUK) using 128-bit AES encryption or stronger where the clear value of the DUK is not accessible in Software.

A CM MUST NOT allow access through Software to any part of the CM device certificate private key.

The CM can do RSA sign and decrypt operations with the CM device certificate private key.

15.1.3 Integrity Protection

Certain parameters are stored in the CM with integrity protection.

The CM MUST store both a) the cable modem private keys and b) public keys and certificates in the trust anchor store inside the CM such that unauthorized modifications are detected using a symmetric keyed hash algorithm which is equivalent to or stronger than HMAC-SHA256 or alternatively using an asymmetric digital signature algorithm which is equivalent to or stronger than 2048-bit RSA.

To generate the integrity protection value, the CM MUST use a cryptographic key that is either the CM device certificate private key or another key which is protected as well as the CM device certificate private key.

The CM MUST provide integrity protection to all of the CableLabs DOCSIS root CA certificate, CableLabs DOCSIS root public key, and the manufacturer name used to validate a CVC attribute.

15.2 Secure Cable Modem Boot

15.2.1 Introduction

The requirements defined in this section address primary security goals for the authentication of Software components at cable modem boot time.

Ensuring the authenticity of the operating system and Software components executing on top of the cable modem is vital to the overall operation of DOCSIS-based networks. Executable Software authentication is achieved by Secure Boot, an attestation process for all Software components residing on the cable modem, ensuring these Software components are verified and authenticated.

15.2.2 Secure Boot

The CM authenticates any new Software downloaded as defined in Section 14, Secure Software Download.

During the next boot process using secure boot mechanisms, the CM MAY either use the same Secure Software Download mechanisms, as defined in Section 14, Secure Software Download (SSD), to validate the Software to be booted or implement other techniques that guarantee the same level of protection.

A CM MUST invoke Secure boot every time it boots and verify the results attest that Software components have not been tampered with as the operating system code is given control.

A CM MUST apply secure boot requirements to all Software component that reside on the CM.

In the case the CM has multiple processors, the CM MUST apply secure boot requirements to all Software components residing on all processors regardless of whether DOCSIS functionality executes on that processor or not.

15.2.3 Chain of Trust

Upon reset, the CM operates secure boot as a chain process where each step verifies the next step in the chain.

The CM MAY implement the secure boot process in such a way that vendors and service providers are be able to sign parts of the Software components for the CM with different cryptographic keys, creating a Chain of Trust.

The CM MAY implement the secure boot process such that signing keys in the chain of trust are different from each other.

The CM signing keys MUST meet the cryptographic requirements that appear in the Code Verification Process section of this specification.

The CM chain of trust MAY allow different entities (e.g., CM OEM, CM ODM and Service Providers) to ensure the integrity of their respective parts of the Software components without sharing signing keys between themselves.

The CM MAY implement a monolithic Software image which represents a simplified case of the chain of trust and multi-processor use cases.

15.2.4 Immutable Root of Trust

An example of immutable root of trust requirements are illustrated in Figure 16.

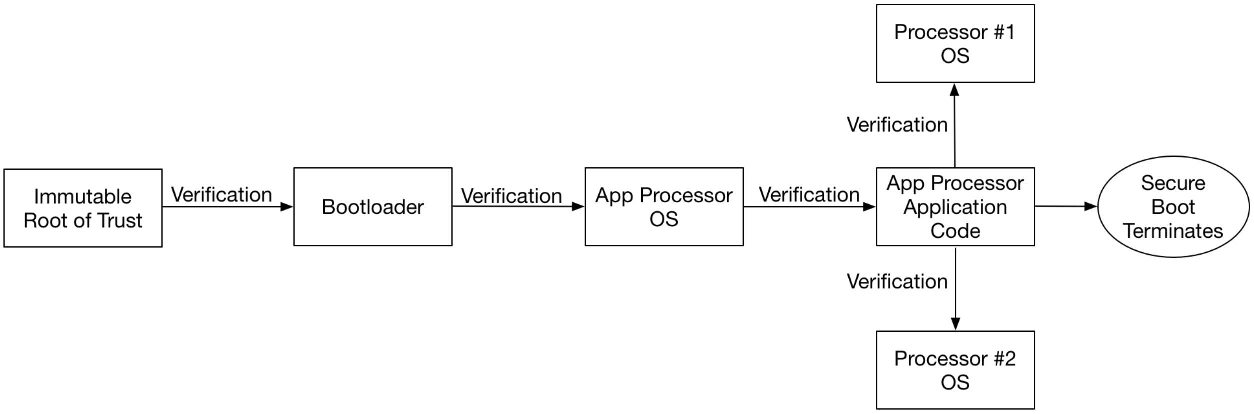

Figure 16 - Immutable Root of Trust Example

The CM Root of Trust MUST be immutable.

All CM cryptographic credentials MUST be immutable.

The CM MUST NOT allow its bootloader code to be overridden or changed.

The CM Root of Trust MAY be unique per manufacture, device model, or network operator.

The CM SHOULD authenticate and verify parameters passed between stages of the secure boot process. For example, the Application Processor OS verifies the authenticity of the root file system, and the root file system includes the OS code that will be loaded into the other processors after authentication.

15.2.5 Code Verification Process

The following requirements apply to the CM code verification process:

- All Software components that are part of the CM Secure Boot requirements MUST be digitally signed.

- The CM MUST use a hashing algorithm equal or stronger to SHA-256 for digitally signing Software components.
- The CM digital signature MUST be done with one of these options: a) RSA Encryption Key modulus that are at least 2048 bits in length, b) elliptic curve encryption with domain parameters that are at least 224 bits in length, or c) industry accepted algorithms with equivalent or stronger security.
- All public keys used for CM code verification MUST be protected for integrity using a code verification key that is either immutable or protected by a chain of keys that starts with an immutable key.
- The key used for digitally signing CM Software components MAY be authenticated using a Code Verification Certificate (CVC).
- Once the operating system starts its operation (after being authenticated by the bootloader) the CM SHOULD verify the Read Only section of the File System that includes the application code and configuration parameters. This is usually referred to as Root File System or Application Code.
- The CM SHOULD implement a continuous validation of the Software components throughout normal operation scenarios. For example, the CM could implement validation of the executable code pages each time they are swapped out of volatile memory and then paged back in.
- If the CM bootloader is composed of multiple separate code modules, each bootloader code module MUST authenticate the next code module that is being authenticated.

Figure 17 shows an example of the secure bootloader process with multiple code modules.

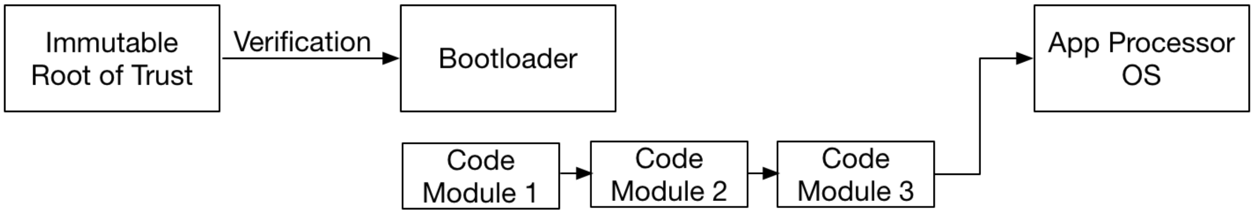

Figure 17 - Immutable Root of Trust

15.2.6 Anti-rollback

During the life of a product, there is a chance of releasing Software components that contain a vulnerability which could potentially impact devices in the field. As part of secure boot, an anti-rollback mechanism is defined to prevent the CM from booting to Software components that are marked as invalid. If the CM does not implement anti-rollback, the vulnerable Software components could be re-installed on the device at some point in the future. Secure Boot Anti-rollback is used to address vulnerabilities that can potentially open up a large population of the deployed units to attacks such as gaining unauthorized root access. Anti-rollback is accomplished through the introduction of a Security Version Number (SVN) that is updated a few times or possibly never during the life of the product. The SVN is not the same as the Software version number that is incremented on every release of the Software image.

15.2.6.1 Security Version Number

A Security Version Number (SVN) is attached to the CM Software components before the Software image is signed for secure boot.

The CM MUST securely record the latest SVN that is part of the latest signed Software image that has been downloaded, installed, and executed on the device.

The CM MUST keep a copy of the SVN in its secure storage.

The CM MUST store the SVN securely such that the CM is able to detect unauthorized manipulation of the SVN at boot up and aborts booting the Software image associated with that stored SVN. Section 15.1 provides requirements on CM parameters which require integrity protection and those requirements apply to the SVN.

At boot time, the CM MUST compare the SVN from the signed image against the SVN kept in its secure storage.

586 At boot time if the image SVN is less than the stored SVN, the CM MUST NOT execute this Software image.

At boot time if the image SVN is equal to the stored SVN and all other code validation steps succeed, the CM MUST execute this Software image.

At boot time if the image SVN is greater than the stored SVN and all other code validation steps succeed, the CM MUST update its stored SVN to the new SVN and execute the Software image.

An SVN is added to the Software image before the image is signed for secure boot. The SVN can be either be embedded into the Software image or appended to the Software image.

The CM MUST calculate the secure boot signature over both the Software image and the SVN.

During secure Software download, the CM MUST compare the SVN from the signed image against the SVN kept in its secure storage.

During secure Software download if the image SVN is less than the stored SVN, the CM MUST abort the Software update process.

Whether the CM implements a single SVN or multiple SVN is outside the scope of this specification. CMs that provide multiple flash banks for Software storage redundancy can assign a separate SVN to each flash bank.

If multiple SVNs are implemented, the CM MUST manage the SVNs appropriately to maintain secure boot.

15.2.6.2 Secure Boot Fail Scenario

The CM MUST re-execute the secure boot procedure in case the secure boot process fails to authenticate any Software component.

The CM MUST NOT complete the boot process if secure boot fails to authenticate a Software component that resides on the CM.

15.3 Secure Shell (SSH) Key Management for Cable Modems

15.3.1 Introduction

Secure Shell (SSH) key management for CMs strengthens remote access to CMs by providing a secure method to provision and manage SSH credentials. The deployment of CM SSH will prevent un-authorized access to CMs on a large scale. In particular, the two options for SSH key (credentials) management are meant to provide a dynamic approach for CM's SSH credentials management: operators can now generate the credentials for specific CMs on the fly and update those credentials every time an access via the SSH interface is needed.

Various use cases are supported by this method including allowing an authorized operator employee to access the CM's SSH server command line interface (CLI). For example, when operators need to gain access to a registered CM, the backend office can push the specific configuration to the CM (e.g., a unique password that is valid for the next 10 minutes to be used on a CM with a specific MAC address) and activate the SSH interface. The technician can now connect via SSH and perform the required operations.

Since an SSH client can use either a password or key for the client authentication, both are included in the list of requirements. Also included are requirements for the SSH server key.

Two different deployment methods are described:

- **Transport Layer Security (TLS)-based Authentication**. This method only works when the CM is online. In particular, allows a CM to authenticate incoming SSH client connections by leveraging a RESTful API secured via a TLS endpoint under the DOCSIS PKI.

- **SNMP-based Authentication**. This method works when the CM is both online and offline. The CM has to register at least once with the CMTS for off-line access. In particular, the authorized set of SSH client public keys and/or passwords needed are provisioned on the CM using configuration file TLVs and/or direct SNMP sets. Once provisioned, the CM uses these authorized keys and/or passwords to authenticate the SSH sessions.

The TLS-based authentication method is optional, and the SNMP-based authentication method is required.

Both methods require that the authorized operator employee acquires an authentication credential (password or public key pair) for their SSH client prior to connecting to the CM's SSH server. The provisioning system then updates its management database with the authorized credentials that are used by the authentication server (TLS-based method) to authenticate SSH client connections or to create signed authorized credential data objects for SNMP configuration.

The CM’s SSH server key can be indirectly authenticated by enabling BPI+ which can be used to authenticate the CM using a digital certificate issued from the CableLabs PKI. Since the SSH server key is located on the CM, that key may be assumed to be valid if the CM is authenticated. The SSH client can also verify the CM’s SSH server key directly, if it knows what keys are authorized, by querying the relevant attributes.

15.3.2 TLS-Based Authentication

A CM MAY implement the TLS-based authentication method as described in this section.

For the TLS-Based authentication method the CM establishes a TLS connection to the authentication server and uses the RESTful API described here to validate the client credentials for SSH access to the CM. The CM-to-Authentication server is mutually authenticated by using the TLS protocol and certificates from the DOCSIS PKI.

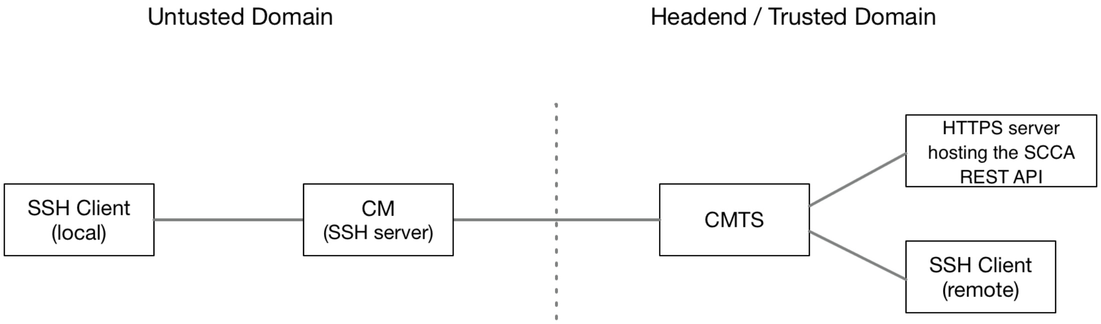

Figure 18 - TLS-based Authentication Architecture

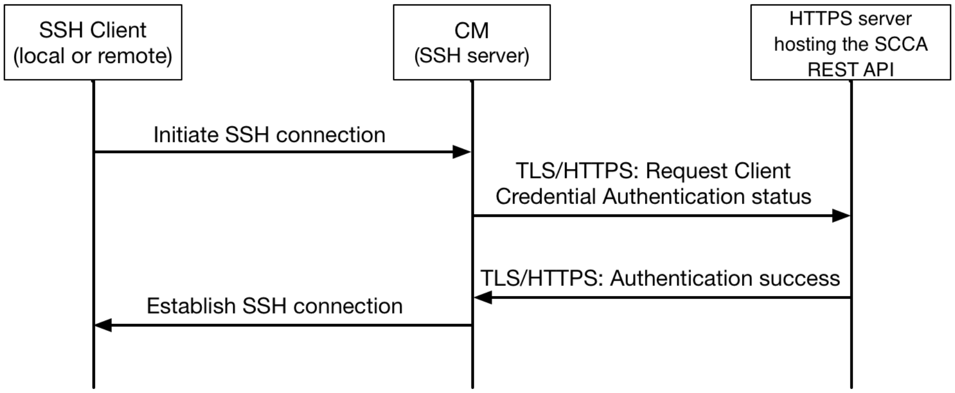
SSH Client (local or remote)
CM (SSH server)
HTTPS server hosting the SCCA REST API
Initiate SSH connection
TLS/HTTPS: Request Client Credential Authentication status
TLS/HTTPS: Authentication success
Establish SSH connection

Figure 19 - TLS-based Message Flow

After the initial setup of the SSH interface on the CM, when a new connection to the SSH interface is detected, the descriptions of the message flow is as follows:

1. The SSH Client initiates a connection to the CM. During the initial protocol exchange, the client presents the authentication credentials (in the form of a username/password combination or a public/private key pair).
2. The CM validates the SSH Client credential by querying the operator's registered authentication endpoint (i.e., the HTTPS server hosting the SCCA REST API). The TLS communication is mutually authenticated (both server and client side) using DOCSIS PKI certificates.
3. If the registered authentication server successfully validates the presented credentials, it indicates to the CM to allow the SSH client to proceed with the connection.

15.3.2.1 SSH Client Credential Authentication (SCCA) Web Service API

The SSH Client Credential Authentication API provides a RESTful interface for CMs to evaluate SSH login credentials directly managed by the operator's authentication server. In particular, the API provides functionality to allow the CM to present client-provided credentials to the registered authentication server which evaluates them and returns the validation status to the CM (i.e., server-side credentials evaluation).

The web service API uses the following URI structure:

<scheme>://<host>:<port>/api/<app>/<version>/<object>

Where:

- <scheme> is the supported protocol scheme.
- <host> is the FQDN or IP address of the HTTPS server hosting the SCCA REST API.
- <port> is the port to use when connecting to the HTTPS server hosting the SCCA REST API. The default value for https is port 443.
- <app> is the name of the application. The value to use for the API described in this document is "ssh-auth".
- <version> is the version of the API. The value to be used for the API described in this document is "1.0".
- <object> is the target data to be accessed.

The CM MUST use a value of "https" for <scheme> in the web service API.

The CM MUST support FQDN or IP addresses for the <host> in the web service API.

The CM MUST use port number 443 in the web service API.

The CM MUST use the value of "ssh-auth" in the <app> field of the web service API.

The CM MUST use the value of "1.0" in the <version> field of the web service API.

The CM MUST populate and use the <object> field in the web service API.

The BASE_URL is defined as the portion of the URI up to, but not including, the resource to be accessed (i.e., the <object>):

BASEURL = <scheme>://<host>[:<port>]/api/ssh-auth/1.0

This section provides a description of the available endpoints and their parameters for the "ssh-auth" application, version 1.0. All endpoint paths are to be considered to be prefixed with the BASEURL described above.

15.3.2.2 Request Messages Contents

All requests sent to the server are of type "application/json". Clients set the "Content-Type" header of any request to "application/json". Clients can set the "Accept" header in the request with the value of "application/json".

15.3.2.3 Response Messages Contents

All responses provided by the server are of type "application/json". The authentication server is required to generate the body of the response with a valid, correctly formatted JSON object with the following structure:

```
{
  “status” : <Integer> ,
  “description” : <string>,
  “data” : <JSON Object>
}
```

Where:

- “status” is the status of the response. The value of the status field will be the same as the HTTP status code used in the response itself.
- “description” is the human-readable description of the status code to be used for logging purposes.
- “data” is the returned data (if any) in JSON format. The structure and fields of the returned body object depend on the associated endpoint.

Table 48 lists status codes that are used for error handling and are common across all the web service API’s endpoints:

Table 48 - Web Service API Status Codes

Code	Status	Description
200	Success	The operation completed successfully. If returned, the “data” field of the JSON object uses base64 encoding.
400	Client Error	The client’s request contained an error that prevented the request to be completed successfully. The “data” field of the returned JSON object is set to “undef”.
403	Forbidden	The CM does not have proper credentials to access the endpoint or the resource. The “data” field of the returned JSON object is set to “undef”.
500	Internal Server Error	The server is in an inconsistent state or does not have the right information to process the request. The “data” field of the returned JSON object is set to “undef”.

The CM MUST both correctly interpret the status codes defined in Table 48 - Web Service API Status Codes and handle other status codes according to HTTP/1.1.

15.3.2.4 Support for Redirection and Security Considerations

In case the HTTPS server hosting the SCCA REST API has been moved, the authentication server can provide the client with a 3XX HTTP redirection response code with a new URL where the resource has been moved. Because of possible security implications, additional considerations are provided for when redirection is used.

The CM can consider the redirected URL to be trusted because the redirect is provided at the HTTP level; since this happens after the TLS session is established, the URL is coming from a trusted entity. Moreover, when the CM connects to the redirected URL the CM initiates a new TLS session, thus requiring a new validation of the server certificate. These conditions guarantee that the redirected endpoint is trusted.

15.3.2.5 HTTPS Server Hosting the SCCA REST API Endpoints

This section provides a description of the defined endpoints and their characteristics. At this time the following endpoint is defined:

Table 49 - Encoding Identifiers for Data Values

Path	Method	Action
/verify	POST	Evaluates the posted credentials and returns the associated login status.

The /verify endpoint is used by the CM to request the authentication server to verify a set of credentials. In particular, when a new connection is requested via the SSH interface, the CM contacts this endpoint to validate them.

If local validation by the CM of the SSH credentials is needed, the SNMP method can be used instead because the SNMP method allows for the Credential Data Structure (CDS) to be downloaded and directly used locally by the CM to validate the SSH login credentials.

15.3.2.5.1 URL Parameters

This endpoint does not require any URL parameters.

15.3.2.5.2 Data Parameters

The data parameters specify the JSON object that is used to carry the credentials to be verified. There are two different types of credentials currently supported. The first one is the 'SSH-Public-Key' that carries the public key presented by the user, while the second is the 'Password' type that carries the user's password. The list and description of the Data Parameters (i.e., the contents of the body of the request) is provided in Table 50:

Table 50 - Fields of the Parameters JSON object

Data Param	Value	Description
id	<userId>	Identifier of the user for which the credentials are to be evaluated by the auth server.
encType	<integer>	Data Encoding for the provided credentials (credValue field). The value is provided to help the authentication server to understand and correctly decode the user credentials. The list of supported encoding types is provided in Table 51.
credType	<integer>	The credType field provides information about the type of credential that is carried in the credValue field. The list of supported types is provided in Table 52.
credValue	Base64(value)	The "credValue" field carries the credential to be verified by the authentication server and is base64 encoded.

The list of supported encoding types for the credentials value (i.e., the `credValue` field) is provided in Table 51.

Table 51 - Encoding Types for Credential Values

ID	Encoding Rules
1	Base64 (B64)
2	RAW
3-127	RESERVED
128-255	VENDOR DEFINED

When using binary values, the CM MUST use the Base64 encoding rules and set the encoding type (i.e., the encType field) to one (1).

> NOTE: When using the `Password` credentials type, the value of the password may be "raw" (i.e., the value of the password itself) or "base64" encoded (i.e., the value of the password is encoded according to the Base64 rules). The encoding type of the credentials has to be set accordingly in the `encType` message property).

There are two different types of credentials currently supported. The first one is the `SSH-Public-Key` that carries the public key presented by the user, while the second is the `Password` type that carries the user's password.

When validating `SSH-Public-Key`, the CM SHOULD set the `credValue` field of the data object to carry the client's public key encoded according to Section 6.6 of [RFC 4253] and Section 3.1 of [RFC 5656] (see 15.3.3.1.4, The Public-Key-Credential Attribute).

When using the `Password` encoding type, please refer to the Password attribute definition in Section 15.3.3.1.1.

The list of supported types of credentials is provided in Table 52:

Table 52 - Identifiers for Credentials Types

ID	Credentials Type
1	SSH-Public-Key
2	Password
3-127	RESERVED
128-255	VENDOR DEFINED

The following is an example data object carrying a password that is provided raw:

```
{ id : userId, encType : 1, credType : 2, credValue : MyPassword }
```

The following is an example data object carrying password that has been Base64 encoded:

```
{ id : userId, encType : 2, credType : 2, credValue : VGhp ... yZAo= }
```

The following is an example data object carrying an SSH-Public-Key:

```
{ id : userId2, encType : 1, credType : 1, credValue : gqhkj ... anGXKTlI= }
```

15.3.2.5.3 HTTP Headers

The endpoint does not require any additional HTTP Headers or HTTP Headers values other than the ones described in 15.3.2.2.

15.3.2.5.4 Responses

The HTTPS server hosting the SCCA REST API constructs the JSON object as defined in Section 15.3.2.3 with the `data` field set to undef. Table 53 provides the list of the valid responses and associated status codes.

Table 53 - Responses and Codes for the /verify endpoint

Code	Status	Description	Data Field Value
200	Success	The operation completed successfully.	undef
403	Not Authorized	The identity is among the configured credentials for the CM, but the passed value was NOT verified correctly.	undef
404	Not Found	The credentials for the CM have been configured, however the requested identity is not present in the set	undef
503	Service Unavailable	The service is not available or credentials for the requesting CM have not been configured on the server.	undef

15.3.2.6 Authentication Server Recommendations

These recommendations are provided for the benefit of the cable operator when implementing the authentication interface for CM's SSH TLS-based client authentication.

The authentication server has to support the SCCA API as described in Section 15.3.2 for receiving and responding to SSH Client credential authentication requests from the CM over HTTPS.

The CM SHOULD use its DOCSIS certificate to authenticate to the server.

15.3.2.6.1 HTTPS Server Hosting the SCCA REST API Configuration

The authentication server has to support TLS version 1.2 as defined in [RFC 5246] or higher for the SCCA REST API HTTPS connections.

For TLS version 1.3, the HTTPS server hosting the SCCA REST API has to support the following ciphers:

- TLS_ECDHE_WITH_AES_128_ SHA256
- TLS_ECDHE_WITH_AES_256_SHA384

For TLS version 1.2, the HTTPS server hosting the SCCA REST API has to support the following TLS cipher suites:

- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384

The implementation of the HTTPS server hosting the SCCA REST API can support other strong TLS cipher suites listed at: https://wiki.mozilla.org/Security/Server_Side_TLS.

The HTTPS server hosting the SCCA REST API has to include its server certificate and the issuing intermediate CA certificate chain in the Certificate message sent to the CM during the TLS message exchange. The HTTPS server hosting the SCCA REST API cannot include the root CA certificate in the TLS Certificate message as defined in [RFC 5246].

The HTTPS server hosting the SCCA REST API has to use a certificate from the DOCSIS PKI to authenticate itself. In particular, the server's certificate has to be a Server-Side certificate usually issued from the Service Provider's Certification Authority, which is available for MSOs from the DOCSIS PKI.

The HTTPS server hosting the SCCA REST API has to request the client to authenticate itself by using its DOCSIS PKI certificate. In particular, the HTTPS server hosting the SCCA REST API has to send a `CertificateRequest` message to the CM during the TLS message exchange and has to close the connection if the CM does not provide a certificate. The HTTPS server hosting the SCCA REST API validates the received CM device Certificate and intermediate CA certificate chain using the Basic Path Validation procedure defined in [RFC 5280]. The CableLabs Root CA certificate for the DOCSIS PKI has to be included in the trust store used for validating client certificates. The HTTPS server hosting the SCCA REST API has to check that the MAC address identifier in the CM device certificate is authorized to request credential authentication status.

The HTTPS server hosting the SCCA REST API could also send the OCSP response for its own (server) certificate and intermediate CA certificate chain to the CM using OCSP Stapling as defined in [RFC 6961].

The HTTPS server hosting the SCCA REST API could check the revocation status for the CM device certificate and its issuing CA certificate by querying the authoritative OCSP responder or by accessing a valid CRL from the issuing CA. The OCSP response `CertStatus` field for each certificate has to have a value of "good" for the process to proceed. If the HTTPS server hosting the SCCA REST API is unable to access the revocation information for the connecting CM, the HTTPS server hosting the SCCA REST API has to be configurable to either not check certificate revocation status or fail CM device certificate validation.

The HTTPS server hosting the SCCA REST API has to support X.509 certificates and validation procedures as defined in [RFC 5280].

The HTTPS server hosting the SCCA REST API has to be able to support the authentication of CMs device certificates (and corresponding issuing CA chains) as defined in Appendix III of this specification.

The certificate of the HTTPS server hosting the SCCA REST API has to use keys sizes of at least 2048 bits for RSA (or equivalent strength if not RSA).

The HTTPS server hosting the SCCA REST API has to use the SHA-256 hashing algorithm (or stronger) for calculating and validating signatures.

15.3.2.7 CM Requirements For TLS-based Authentication

The CM MUST disable SSH server functions by default.

The CM MUST support the Attributes defined in Section 15.3.3, SNMP-based Authentication.

If the URL for the HTTPS server hosting the SCCA REST API has not been configured on the CM, the CM MUST NOT use the TLS-based authentication method.

If the URL for the HTTPS server hosting the SCCA REST API has been configured on the CM, the CM MUST use the TLS-Based Authentication method for checking SSH client credentials with the authentication server.

When an SSH client initiates an SSH CLI connection to a CM, the CM MUST check the authentication status of the presented credential by sending a request to the authentication server by using the SCCA API as described in Section 15.3.2, TLS-Based Authentication.

The CM MAY support checking the authentication status of an SSH client's public key credential and a PAM module could be used to implement this functionality.

The CM MUST NOT allow an SSH client to access its CLI until it receives a "success"/good response from the authentication server via the SCCA API.

The CM MUST include its CM device certificate and intermediate CA certificate chain (from the DOCSIS PKI) in the Certificate message of the HTTPS TLS message exchange for the SCCA API connection.

The CM MUST validate the server certificate and intermediate CA certificate chain during the HTTPS TLS message exchange using the Basic Path Validation procedure defined in [RFC 5280].

The CM MUST use the CableLabs Trust Anchors (or Root CAs) from the DOCSIS PKI as the trust anchor to validate the identity of the SCCA API web services interface.

The CM MUST verify that the domain name identifier in the authentication server's certificate matches the domain name of the SCCA API URL when the URL host is a FQDN.

The CM SHOULD check the revocation status for the authentication server certificate and its intermediate CA certificate chain by processing the OCSP responses either via OCSP Stapling during the TLS negotiation (as defined in [RFC 6961]) or by directly querying the authoritative OCSP responders.

The CM MUST verify that the CertStatus field for each certificate has a value of "good" for every certificate in the server's certification chain (excluding the Root CA).

If the HTTPS server hosting the SCCA REST API is not able to provide its own revocation status to the CM (e.g. OCSP server cannot be reached), the CM can either continue operation or reject the TLS connection depending upon the CM's configuration. If not explicitly prohibited, the default behavior is to continue with the connection.

The CM MAY cache OCSP responses that have not expired for the duration of their validity period.

The CM MUST support only SSH-2 for SSH connections as described in [RFC 4253].

The CM MUST deny new SSH connections after a configurable amount of time from when the interface has been enabled (default is 0 with a maximum of 8 hours) (see Section A.1.2.1.1, SSH New Connection Timeout).

The CM MUST disable existing SSH connection if there has been no activity for a configurable amount of time (default is 30 minutes with a maximum of 24 hours) (see Section A.1.2.1.1, SSH New Connection Timeout).

The CM MUST support X.509 certificates as defined in [RFC 5280].

The CM MUST support CM device certificate and issuing CA chain profiles as defined the Appendix III of this specification.

The CM MUST support the use of private keys of at least 2048-bits for RSA (or equivalent security bits).

The CM MUST support the validation of RSA signatures generated with keys of up to 8192-bits.

The CM MUST support the SHA-256 hashing function (or stronger) for certificate signatures.

The CM MUST support TLS version 1.2 as defined in [RFC 5246] or higher for connecting to the HTTPS server hosting the SCCA REST API.

The CM MUST support, at minimum, the following TLS cipher suites:

- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384

The CM MAY support the following TLS cipher suites:

- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384

Additionally, the CM MAY support other strong TLS cipher suites (see "Intermediate Compatibility" list from https://wiki.mozilla.org/Security/Server_Side_TLS).

15.3.3 SNMP-based Authentication

The CM MUST implement the SNMP-based Authentication method as defined in this section.

The SNMP-Based Authentication method uses SNMP to configure an authorized set of SSH Client credentials in the CM. The CM uses these credentials to authenticate client connections for accessing its command line interface.

The main architecture, depicted in Figure 20, consists of a provisioning system, an SSH Client, and a CM. The provisioning system includes an SNMP manager and may also include a file server for downloading the credentials from. The CM exposes an SSH interface for supporting access to its CLI subsystem.

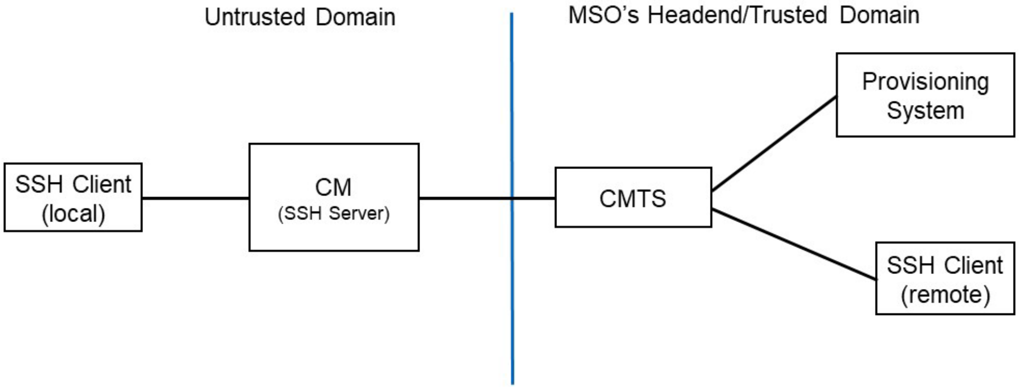

Figure 20 - SNMP-Based Authentication Architecture

In order to enable CLI access to the CM, the operator configures the SSH server on the CM by providing either (a) a URL that the CM uses to download an authorized Credential Data Structure (CDS) (pull model), or (b) the CDS itself (push model). If the CDS is too large, it is recommended to provide a URL to the CM to download the CDS since the SNMP MIB may not be able to support a direct configuration of that size a file.

The configuration options for the CM’s SSH server are provisioned via TLVs in the config file or by setting attributes via an authenticated management system, e.g. SNMPv3.

The CDS contains:

- one or more authorized set of credentials in the form of public keys or username/password combinations.
- (optionally) a CM MAC address qualifier to associate a credential to a particular CM.
- (optionally) a digital signature that authenticates the contents of the CDS

The CDS is created or updated when (or before) credentials are provided to operator personal that need CM CLI access.

Because the CDS can contain bare credentials (e.g., passwords), it is important that the CDS is secured in transit to the CM.

When the HTTPS mechanism is used, the CM MUST verify the server’s certificate by using the DOCSIS PKI’s Root CA as the trust anchor to verify that the CDS is provisioned from a trusted server.

BPI+ Early Authentication and Encryption (EAE) is recommended to be enabled, especially when using the SNMP-based authentication method. When BPI+ and EAE are enabled, the CM requesting a CDS download can be assumed trusted since the CMTS verifies, at a minimum, that the requesting entity is a valid CM. The CM’s device certificate is also used to authenticate the CM when using an HTTPS connection. BPI+ encryption also protects SNMP traffic over the DOCSIS network when directly configuring the CDS.

> NOTE WELL: When using the CDS to deploy authorized passwords, the use of shared passwords across multiple CMs (i.e., it does not have a MAC address qualifier) can compromise all CMs that share the same CDS. For example, an attacker could attack and gain entry to a CM, get the SSH password, and then use that password to gain access to other CMs that have a CDS file with the same authorized password. For these reasons, it is highly discouraged to deploy the same credentials across different devices and, if strictly needed, it is suggested that (very) limited validity periods are used to limit any possible exposure.

The SSH credentials needed by the CM to authenticate inbound connections are transferred via the CDS which can be both authenticated and encrypted. When the CDS is transferred via an insecure method (i.e., without encryption and/or authentication), the CDS is recommended to be encrypted using a Cryptographic Message Syntax (CMS) Enveloped-Data structure as defined in [RFC 5652]. The encrypted CDS is then transferred via the Encrypted-CDS attribute – in other words, the CDS data is encrypted for the public key in the CM's certificate which can be easily evaluated when BPI+ is enabled.

Ultimately, the status of the SSH server on the CM is provided via the SshCmServerStatusCode as described in [DOCSIS CM-OSSIv4.0].

15.3.3.1 The Credential Data Structure (CDS)

The CDS is a compound attribute, consisting of one or more sub-attributes.

The Credential Data Structure (or CDS) provides one or more sets of credentials for SSH incoming connection authentication.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|  Type = 1     |            Length             | compound  ...
+---------------+---------------+---------------+---------------+
```

compound

The CDS attributes are listed in Table 54.

The CDS can contain one or more Password-Credential attributes, one or more Public-Key-Credential attributes, and, optionally, one and only one CDS-Signature attribute.

Table 54 - Credential Data Structure Attributes

Attribute	Contents	See
Password-Credential	Username and Password combination credentials	Section 15.3.3.1.1
User-ID	Username	Section 15.3.3.1.2
User-Password	User password associated with the username	Section 15.3.3.1.3
Public-Key-Credential	RSA Public Key Credential	Section 15.3.3.1.4
SSH-Public-Key	Public key	Section 15.3.3.1.5
CDS-Signature	Detached CMS signature for CDS authentication	Section 15.3.3.1.6

15.3.3.1.1 The Password-Credential Attribute

The Password-Credential is a compound attribute that contains a username and password combination.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|  Type = 3     |            Length             | compound  ...
+---------------+---------------+---------------+---------------+
```

compound

This attribute carries:

- one and only one `User-ID` attribute
- one and only one `User-Password` attribute
- (optionally) one or more `MAC-Address` attribute(s)

When `MAC-Address` sub-attributes are present, the CM MUST allow the use of the credential only on the interface whose MAC address matches the `MAC-Address` attribute's value.

The Password-Credential attribute consists of the sub-attributes described in Table 55.

Table 55 - Password-Credential Attributes

Attribute	Contents	See
User-ID	User ID	Section 15.3.3.1.1
User-Password	User password associated with the username	Section 15.3.3.1.3
MAC-Address	MAC address of the target CM where to install the credential.	Section 7.2.2.3

15.3.3.1.2 The User-ID Attribute

The User-ID attribute contains the Username (or identifier) connected to the value of the `User-Password` carried in the `Password-Credential` compound attribute.

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +---------------+---------------+---------------+---------------+
 |  Type = 4     |        3 < Length < 256       | string    ...
 +---------------+---------------+---------------+---------------+
```

`string`

The User-ID string field is at least three octets and has a maximum length of 255 octets.

The CM MUST only accept a User-ID encoded in the [ISO 8859-1] character-set and using only characters in the range from 0x21 – 0x7E.

15.3.3.1.3 The User-Password Attribute

The User-Password attribute contains the password associated with the User-ID.

```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +---------------+---------------+---------------+---------------+
 |  Type = 5     |        5 < Length < 256       | string    ...
 +---------------+---------------+---------------+---------------+
```

`string`

The User-Password string field is at least five octets and has a maximum length of 255 octets. The suggested range for the field's length is 12 to 32 octets (See Section 15.3.4.1).

The CM MUST only accept a password encoded in the [ISO 8859-1] character-set and using characters in the range from 0x21 – 0x7E. Recommendations for generating the User-Password are in Section 15.3.4.1.

15.3.3.1.4 The Public-Key-Credential Attribute

This is a compound attribute that contains a public key and an optional MAC-Address sub-attribute.

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +---------------+---------------+---------------+----------------+
   |  Type = 6     |             Length            | compound   ...
   +---------------+---------------+---------------+----------------+
```

`compound`

This attribute contains only one of `Public-Key` sub-attribute and an optional MAC-Address sub-attribute.

When the `MAC-Address` sub-attribute is present, the CM MUST allow the use of the credential only on the interface whose MAC address matches the `MAC-Address` sub-attribute value.

The list of sub-attributes is provided in Table 56.

Table 56 - Public-Key attributes

Attribute	Contents	See
SSH-Public-Key	An SSH public key data structure that carries the value of the public key and associated parameters.	Section 15.3.3.1.5
MAC-Address	MAC-Address of the target CM where to install the credential.	Section 7.2.2.3
Message-Signature	Detached CMS signature for CDS authentication	Section 15.3.3.1.6

15.3.3.1.5 The SSH-Public-Key Attribute

The SSH-Public-Key attribute carries the value of a public key.

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +---------------+---------------+---------------+---------------+
   |  Type = 7     |             Length            | string   ...
   +---------------+---------------+---------------+---------------+
```

`string`

The value of this attribute carries a public key encoded according to Section 6.6 of [RFC 4253] and Section 3.1 of [RFC 5656].

The CM MUST support both RSA and ECDSA public keys types.

The CM MUST support RSA public keys with modulus lengths of 2048, 3072, and 4096 bits.

For ECDSA public keys, the CM MUST support the nistp256, nistp384, and nistp521 curves as described in [RFC 5656] Section 10.1.

15.3.3.1.6 CDS-Signature

The CDS-Signature attribute carries the authentication of the entire CDS, excluding the CDS-Signature attribute.

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +---------------+---------------+---------------+---------------+
   |  Type = 8     |             Length            | string   ...
   +---------------+---------------+---------------+---------------+
```

`string`

The attribute value carries a detached CMS data structure with the Signed-Data as defined in Section 5 of [RFC 5652]. The signature is calculated over all the CDS sub-attributes, excluding the CDS-Signature attribute itself, in the same order in which the attributes appear within the compound attribute.

When the CDS is signed, the CM MUST verify both the signature and the certificate chain to be anchored to the DOCSIS Root CA before using the CDS.

15.3.3.2 The Encrypted-CDS Attribute

The Encrypted Credential Data Structure (or Encrypted-CDS) is a simple attribute used to transfer an encrypted CDS to the CM using insecure channels. When no secure transfer method is available (e.g., no HTTPS server hosting the SCCA REST API, SNMP authentication, or BPI+ encryption enabled), the CDS has to be provided in an encrypted CMS structure. In this case, the carried CDS structure has to contain the Message-Signature attribute where the signing certificate has to be from the DOCSIS PKI.

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +---------------+---------------+---------------+---------------+
   |  Type = 2     |             Length            | string   ...
   +---------------+---------------+---------------+---------------+
```

`string`

The Encrypted-CDS value contains a CMS (Enveloped-Data) as defined in [RFC 5652] containing the CDS compound attribute defined in Section 15.3.3.1. The content encryption key has to be generated and encrypted using the “key agreement” technique defined in [RFC 5652] with the Public Key from the CM’s device certificate (DOCSIS PKI). The CM's device certificate (and the issuing CA's certificate) can be accessed via SNMP and properly validated before encrypting the credentials.

When no secure transfer method is available, additional validation steps can be taken to ensure that the CDS is delivered to an authorized cable modem and not to a previously compromised cable modem being used to steal SSH credentials. This additional validation can be done by:

- checking the MAC address of the cable modem about to be provisioned with SSH credentials against known compromised cable modem MAC addresses,
- a back-office system can use SNMP and standard DOCSIS MIBs to verify that the cable modem MAC address is authorized on a CMTS and that the cable modem is currently registered and active with a valid authorization association.

  NOTE: This form is not appropriate for shared CDS use cases because the CDS can only be encrypted for a single CM. In cases where the same CDS is provided to multiple cable modems (via a URI), session-based encryption (e.g., via TLS) could be used or each CM could be provided its own CDS.

When the CMS is encrypted, the CM MUST successfully decrypt the CMS by using its own DOCSIS device certificate (and private key) before using the decrypted CDS to authenticate SSH client connections.

15.3.3.3 Provisioning System Recommendations

These recommendations are provided for the benefit of the cable operator when implementing a provisioning system that supports cable modem SSH SNMP-based authentication.

The provisioning system has to support HTTPS for downloading the CDS.

The provisioning system has to support TLSv1.2 as defined in [RFC 5246] and could support TLSv1.3.

The HTTPS endpoint used to download the CDS could support OCSP Stapling as defined in [RFC 6961].

For TLS version 1.3, the TLS interface of the provisioning system has to support the following cipher suites:

- TLS_ECDHE_WITH_AES_128_ SHA256
- TLS_ECDHE_WITH_AES_256_SHA384

For TLS version 1.2, the TLS interface of the provisioning system has to support the following cipher suites:

- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384

Provisioning system implementations are encouraged to track developments in strong TLS cipher suites, specifically the "Intermediate Compatibility" list at: https://wiki.mozilla.org/Security/Server_Side_TLS

The provisioning system has to support signing the CDS as defined in Section 15.3.3.1.6 when the CDS is transferred via insecure protocols (e.g., un-authenticated SNMP).

The provisioning system has to support encrypting the CDS as defined in 15.3.3.1.6 when the CDS is transferred via insecure protocols and the CDS contains secrets (e.g., non-public-key based credentials).

15.3.3.4 CM Requirements For SNMP-based Authentication

The CM MUST support the setting of the SSH Server Settings (described in Section 15.3.4, CM SSH Server Configuration File TLVs) via SNMPv3.

The CM MUST enable the SSH server as indicated by the SSH configuration settings.

If SshCmCds or SshCmCdsDownloadUrl TLVs are present in the config file or the SSH New Connection Timeout has not been set to non-zero values (i.e., 1 ... 28800), the CM MUST disable all SSH interfaces.

The CM MUST support removal of the current CDS by setting a 0-length CDS via SNMPv3.

The CM MUST allow ongoing SSH connections to remain open, subject to the SSH Inactivity Timer.

The CM MUST attempt to download the CDS using the URL when the CDS download URL is properly configured either via config file or via SNMP.

After the CDS has been successfully verified, the CM MUST install the relevant credentials contained in the CDS and enable them for SSH client connections.

The CM MUST enable the SSH interfaces and accept inbound connections when SSHNewConnectionTimeout timer is set to a non-zero values and there is one or more credentials installed via a CDS.

The CM MUST stop accepting new SSH connections and destroy the CDS after the SSHNewConnectionTimeout expires.

The CM MUST keep established SSH connections active when/after the SSHNewConnectionTimeout expires.

The CM MUST verify that SSH Client credential provided during connection establishment (public key or user/password) exists in the CDS and that the MAC-Address specifier (when present) matches the activated interface’s MAC address before establishing an SSH session.

The CM MUST support SSH-2 only for SSH connections as described in [RFC 4253].

The CM trust anchor store MUST include the CableLabs Root CA certificate for validating provisioning system certificates when using HTTPS.

When HTTPS is used to download the CDS from the provisioning system, the CM MUST include its CM device certificate and intermediate CA certificate chain during the TLS negotiation.

When HTTPS is used to download the CDS from the provisioning system, the CM MUST validate the server’s certificate and its intermediate certification chain during the TLS negotiation using the Basic Path Validation procedure defined in [RFC 5280].

When HTTPS is used to download the CDS from the provisioning system, the CM MUST use the CableLabs Root CA certificate as the trust anchor for the server’s certificate validation.

When HTTPS is used to download the CDS, the HTTPS server has to request the client to authenticate itself by using its DOCSIS PKI certificate. In particular, the server has to send a `CertificateRequest` message to the CM during the TLS message exchange and has to close the connection if the CM does not provide a certificate. The server has to validate the received CM device Certificate and intermediate CA certificate chain using the Basic Path Validation procedure defined in [RFC 5280]. The CableLabs Root CA certificate for the DOCSIS PKI has to be included in the trust store used for validating client certificates.

When HTTPS is used to download the CDS from the provisioning system, the CM SHOULD check the revocation status for the server certificate and its certification chain by processing valid CRLs or OCSP responses received via OCSP stapling [RFC 6961] or by directly contacting the authoritative servers.

When HTTPS is used to download the CDS from the provisioning system, the CM MUST verify that the CertStatus field for each OCSP response for the server's certification chain has a value of 'good'.

Alternatively, the CM MAY download, for every certificate in the server's certification chain, the relevant CRLs and verify that the certificates have not been revoked (i.e., they are not listed in the CRL).

When HTTPS is used to download the CDS from the provisioning system and the provisioning system server is not able to provide revocation status to the CM (e.g., OCSP server is down), the CM MUST continue operation or reject the connection depending upon the SshCmRevStatusUnavailable attribute configuration (default = continue operation).

When HTTPS is used to download the CDS from the provisioning system, the CM MAY cache OCSP responses and/or CRLs for their entire validity period.

The CM MUST support X.509 certificates as defined in [RFC 5280] and the CM device certificate and issuing CA chain profiles defined in Appendix III, Certification Authority and Provisioning Guidelines (Informative) of this specification.

The CM MUST support SHA-256 HASH algorithm for both signature validation and signature generation for the CDS.

The CM MAY support stronger hashing algorithms (e.g., SHA-384 or SHA-512).

The CM MUST support TLS version 1.2 [RFC 5246] or higher for connecting to TLS protected endpoints.

The CM MUST support the following TLSv1.2 cipher suites for the HTTPS connection used to download the CDS:

- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384

In case TLSv1.3 is supported by the CM, the CM MUST support the following TLSv1.3 cipher suites:

- TLS_ECDHE_WITH_AES_128_SHA256
- TLS_ECDHE_WITH_AES_256_SHA384

Implementations are encouraged to keep up to date with the list of TLS recommendations provided at: https://wiki.mozilla.org/Security/Server_Side_TLS.

15.3.4 CM SSH Server Configuration File TLVs

Annex A describes the configuration file settings for the CM SSH server. The values can be set via configuration file or authenticated management protocols such as SNMPv3.

15.3.4.1 Credential Recommendations for Providing CLI Access to CMs

For this service to meet current security norms, the following recommendations are made.

It is recommended that both the client key and password:

- be unique to each device and not per user.
- be short-lived with a minimum of 7 days and a maximum of one month.
- be renewable where the credential is not compromised with the credential update protected with an encryption key that is at least as strong as the credential itself. If this update is done over the air, the device is to be authenticated using its DOCSIS certificate or a credential of equivalent strength.

It is recommended that the client key be RSA 2048-bit key equivalent or stronger.

It is recommended that the client password:

- be random as defined in [NIST SP800-90A].
- not be derivable using a weak or a common (shared) key as seed. (For example: deriving a unique password using device MAC address and a common or hardcoded key.)
- be at least 16 characters long.
- be generated based on recommended guidelines for the character set in [NIST SP800-63B].

It is recommended that the server credential:

- be RSA 2048-bit key equivalent or stronger
- be unique to each device
- have the same physical security requirements for the device private key as are applied to the SSH credentials.

It is recommended that the client authenticate the server public key before it trusts and stores the public key in its trusted key storage.

15.4 CMTS Management Security Guidelines

The requirements listed here provide guidelines for initial CMTS security settings and defaults when shipped from the manufacturer to an operator.

The requirements listed here provide default configuration for either a traditional CMTS or CCAP Core, that is, CMTS equipment typically installed in a headend or distribution hub and does not include DAA components installed in the outside plant or virtualized CCAP components deploying NFV.

15.4.1 DOCSIS 4.0 CMTS Networking Defaults

The following CMTS networking defaults apply to a CMTS shipped from a supplier to an operator. Networking services such as those listed in this section are not allowed to be configured until a system password or credential has been set by an authorized operator employee.

The CMTS by default SHOULD NOT be available on a network except for local dedicated management ports until that CMTS has been configured.

The CMTS by default MUST disable DHCP, TOD, TFTP, IPDR and Syslog services until server addresses are configured.

The CMTS by default SHOULD NOT have any IP addresses, ports, or routes configured.

The CMTS by default SHOULD NOT connect to network time services.

The CMTS by default MUST disable remote management and remote access protocols such as HTTP, HTTPS, SNMP SSH, and telnet.

The CMTS by default MUST support TLS v1.2.

The CMTS by default SHOULD support TLS v1.3.

The CMTS by default MUST support SSHv2.

The CMTS by default MUST support the following TLS cipher suites:

- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384

The CMTS by default MAY support other strong TLS cipher from the "Intermediate Compatibility" list from https://wiki.mozilla.org/Security/Server_Side_TLS).

The CMTS MUST support disabling cipher suites. An operator can choose to disable cipher suites they deem not secure.

The CMTS by default SHOULD NOT have default values for the system group in MIB2.

The CMTS by default SHOULD NOT have a hostname populated.

CMTS implementations are encouraged to track developments TLS cipher suites from the "Intermediate Compatibility" list at: https://wiki.mozilla.org/Security/Server_Side_TLS.

CMTS implementations are encouraged to track SSH crypto algorithms in both “Modern” or “Intermediate” lists at: https://infosec.mozilla.org/guidelines/openssh.html.

15.4.2 DOCSIS 4.0 CMTS Initial Configuration

Default credentials are to be avoided whenever possible.

The CMTS SHOULD be left in a secure configuration, including the prevention of unauthorized access, by any defaults not explicitly listed.

The CMTS MUST prompt to change both username and the associated secret at the first login attempt if default credentials were used.

First access to the CMTS MUST be only via a local physical console or management port.

When first accessing the CMTS, the CMTS MUST prompt the user to set a login credential.

When first accessing the CMTS and after setting the login credential, the CMTS SHOULD default to an internal setup application that provides configuration options including options to guide the user through first time configuration of the device, an option to download a configuration file, or manually configure the device.

15.4.3 CMTS DOCSIS Security Default Settings

The following DOCSIS services are either not provisioned or left empty until explicitly configured for the first time.

There are default CMTS requirements listed in the following sections of this specification.

- Section 8.4 states that by default BPI+ Early Authentication and Enforcement (EAE) uses Ranging-Based EAE Enforcement (Policy 2).
- Section 9.4.2.4 states by default the CMTS enables configuration file learning.
- Section 9.4.2.5 lists defaults for CMTS TFTP options.
- Section 9.6 states that by default the CMTS has Source Address Verify (SAV) on.
- Annex A lists default BPI+ configuration parameters.

The CMTS by default MUST NOT have a shared secret provisioned.

The CMTS by default MUST NOT have the DOCSIS DHCP Relay function enabled.

The CMTS by default MUST have an empty BPI+ EAE exclusion list.

The CMTS by default MUST NOT have RF interfaces configured.

The CMTS by default MUST NOT have any MAC domains configured.

The CMTS by default MUST only have the DOCSIS Root CA certificate and CableLabs Root CA certificate configured as trust anchors for validating CM device certificates during BPI+ authentication.

The CMTS by default MUST NOT have any sub-CA or any intermediate CA certificates configured.

The CMTS by default MUST NOT have any CM device certificates configured on the exclusion list.

15.4.4 Restore Factory Defaults

In the lifecycle of a CMTS there are times when restoring the CMTS to factor defaults is needed, e.g., decommissioning. Restoring to factory defaults includes providing a method to remove configuration information, credentials, secret values, and other administrative data.

The CMTS MUST support a method to restore the CMTS to its initial factory default configuration.

Annex A TFTP Configuration File Extensions (Normative)

A CM's security configuration parameters are included in the CM configuration file that is downloaded from a TFTP server [DOCSIS MULPIv4.0].

A.1 Encodings

The following type/length/value encodings are used for security configuration settings included in the configuration file. The security configuration settings in the RF MAC CM registration requests MUST be bitwise identical to those included in the configuration file. All multi-octet quantities are in network-byte order, i.e., the octet containing the most-significant bits is the first transmitted on the wire.

A.1.1 Baseline Privacy Plus Configuration Setting

The combination of Privacy Enable configuration setting [DOCSIS MULPIv4.0] and the Privacy Support Modem Capability Setting [DOCSIS MULPIv4.0] controls whether Baseline Privacy Plus is enabled or disabled in a CM. If the configuration file does not contain all the necessary BPI+ parameters, the CM MUST use the default value(s) specified in Table 57 for the missing parameter(s). The separate Privacy Enable parameter allows an operator to disable or re-enable Baseline Privacy by toggling a single configuration parameter.

This field defines the parameters associated with Baseline Privacy operation. It is composed of a number of encapsulated type/length/value fields. The type fields defined are only valid within the encapsulated Baseline Privacy configuration setting string.

Type	Length	Value
BP_CFG	n	

[DOCSIS MULPIv4.0] defines the specific value of BP_CFG.

A.1.1.1 Internal Baseline Privacy Encodings

A.1.1.1.1 Authorize Wait Timeout

The value of this field specifies the retransmission interval, in seconds, of Authorization Request messages when in the Authorize Wait state.

Subtype	Length	Minimum Value	Maximum Value
1	4	1	30

A.1.1.1.2 Reauthorize Wait Timeout

The value of this field specifies the retransmission interval, in seconds, of Authorization Request messages when in the ReAuthorize Wait state.

Subtype	Length	Minimum Value	Maximum Value
2	4	1	30

A.1.1.1.3 Authorization Grace Time

The value of this field specifies the grace period for reauthorization, in seconds.

Subtype	Length	Minimum Value	Maximum Value
3	4	1	6,047,999

A.1.1.1.4 Operational Wait Timeout

The value of this field specifies the retransmission interval, in seconds, of Key Requests when in the Operational Wait state.

Subtype	Length	Minimum Value	Maximum Value
4	4	1	10

A.1.1.1.5 Rekey Wait Timeout

The value of this field specifies the retransmission interval, in seconds, of Key Requests when in the Rekey Wait state.

Subtype	Length	Minimum Value	Maximum Value
5	4	1	10

A.1.1.1.6 TEK Grace Time

The value of this field specifies grace period, in seconds, for re-keying the TEK.

Subtype	Length	Minimum Value	Maximum Value
6	4	1	302,399

A.1.1.1.7 Authorize Reject Wait Timeout

The value of this field specifies how long, in seconds, a CM waits in the Authorize Reject Wait state after receiving an Authorization Reject.

Subtype	Length	Minimum Value	Maximum Value
7	4	1	600

A.1.1.1.8 SA Map Wait Timeout

The value of this field specifies the retransmission interval, in seconds, of SA Map Requests when in the Map Wait state.

Subtype	Length	Minimum Value	Maximum Value
8	4	1	10

A.1.1.1.9 SA Map Max Retries

The value of this field specifies the maximum number of Map Request retries.

Subtype	Length	Minimum Value	Maximum Value
9	4	0	10

A.1.2 CM SSH Server Configuration Settings

This section describes configuration file settings for the CM SSH server described in Section 15.3. The values can be set via configuration file or authenticated management protocols such as SNMPv3.

The CM MUST apply these settings upon enabling the physical interfaces when an SSH Server Settings TLV is present in the CM configuration file.

he CM MUST apply the default settings described in the TLVs when an SSH Server Settings TLV is not present in the CM configuration file.

A.1.2.1 Common SSH Configuration File Settings

These configuration file settings are common to both the TLS and SNMP methods for CM SSH access.

Type	Length	Value
98.1.x	n	

A.1.2.1.1 SSH New Connection Timeout

This field indicates the amount of time the SSH interfaces will be accessible on the CM. When the SSH New Connection Timeout TLV is set to non-zero, the CM MUST activate the SSH interfaces according to the configured SSH attributes.

Type	Length	Value
98.1.1	4	New connection timeout (in seconds) (0-28800, 0: SSH disabled)

The CM MUST listen for new SSH inbound connections for the specified amount of time (in seconds) from the moment the SSH New Connection Timeout is set.

The CM MUST disable all SSH interfaces when the SSH New Connection Timeout expires.

The CM MUST disable all SSH interfaces when the SSH New Connection Timeout value is set to 0.

The CM MUST allow any SSH sessions opened prior to the expiration of the SSH New Connection Timeout to remain active (i.e., the timeout doesn’t apply to already-established SSH sessions).

The CM MUST use a default value of 0 (SSH interfaces disabled) for the SSH New Connection Timeout.

The maximum value for the SSH New Connection Timeout timer is 28800 seconds (equivalent to 8 hours).

A.1.2.1.2 SSH Inactivity Timeout

This field indicates the amount of time (in seconds) of user inactivity before an SSH session is terminated.

Type	Length	Value
98.1.2	4	Inactivity timeout (in seconds) (0-86400, 0: Disabled)

A CM MUST close any SSH session after the designated number of seconds of user inactivity in the SSH Inactivity Timeout timer (i.e., no keystrokes have been entered in the SSH session for the given number of seconds).

The CM MUST use a default value of 1800 (30 minutes) for the SSH Inactivity Timeout.

A.1.2.1.3 SSH Enabled Interfaces

This bitmask field indicates which interfaces the SSH server is enabled on.

Type	Length	Value
98.1.3	1	Enabled Interfaces Bitmap bit #0: All local (customer premises) network interfaces/addresses. This includes Ethernet, wireless and MOCA interfaces. bit #1: All network-facing private interfaces/addresses (i.e. the operator network) bit #2-7: reserved.

The CM MUST use a default value of 0x02 for the SSH Enabled Interfaces bitmask (i.e., only network-facing private interfaces are enabled).

Bits 2 through 7 of the SSH Enabled Interfaces bitmask are reserved for future use and are ignored.

When the SSH Enabled Interfaces TLV is set, CM MUST enable SSH access only on the network interfaces (and associated IP addresses) that have their corresponding bit set (1) in the SSH Enabled Interfaces bitmask attribute.

The CM MUST disable SSH access for all the interfaces (and associated IP addresses) whose corresponding bit in the SSH Enabled Interfaces bitmask attribute is not set (0).

The CM MUST not close any open SSH sessions when the SSH Enabled Interfaces value is changed (i.e., existing connections are not affected by this attribute).

A.1.2.1.4 SSH Source Address Restrictions

A network/address specifier in CIDR notation that limits the IP addresses where SSH connections can originate.

Type	Length	Value
98.1.4	n	Network/address specifier (in CIDR notation)

The CM MUST only allow SSH connections if the source address in the SSH Source Address Restrictions TLV matches the network/address specifier.

For example, a network/address specifier of "10.10.0.0/16" would allow SSH connections from "10.10.1.25" but would not allow connections from the host with IPv4 address 10.20.1.25. And a CIDR of "10.10.1.2/32" would only allow SSH connections from the host with IPv4 address 10.10.1.2.

The CM MUST NOT close any open SSH sessions when the SSH Source Address Restrictions value is changed.

The CM MUST enable unrestricted source address access to the SSH server when the SSH Source Address Restrictions TLV is not present in the CM config file.

A.1.2.1.5 SSH SCCA Certificate Revocation Check Disable

This field indicates if certificate revocation checks for the SCCA TLS connection are disabled.

Type	Length	Value
98.1.5	1	If set to non-zero value, the CM proceeds even without rev info (0: do not proceed)

The CM MUST check the revocation status of the chain of certificates that authenticate the authentication server when connecting via TLS.

If the value of the SSH SCCA Certificate Revocation Check Disable TLV is set to 0 (default), then the CM MUST NOT proceed with the authentication of the server if the revocation information is missing or otherwise not available.

When the value of the SSH SCCA Certificate Revocation Check Disable TLV is set to non-zero (i.e., 1-255), the CM MUST proceed with the server's credentials validation even when no valid revocation information is available for the server's credentials.

A.1.2.2 TLS-based Authentication Configuration Options

These configuration file settings are used for the TLS method for CM SSH access.

Type	Length	Value
98.2.x		

A.1.2.2.1 SSH SCCA REST API URL

This field indicates a URL where the SCCA REST API is implemented.

Type	Length	Value
98.2.1	n	SCCA REST API URL endpoint

When using the TLS-based authentication the CM MUST invoke endpoints on the URL described in the SSH SCCA REST API URL TLV to validate the credentials presented by the client (see Section 15.3.2.6.1, HTTPS Server Hosting the SCCA REST API Configuration).

The CM MUST support HTTPS URLs.

A.1.2.3 SNMP-based Authentication Configuration Options

These configuration file settings are used for the SNMP method for CM SSH access.

Type	Length	Value
98.3.x		

A.1.2.3.1 SshCmCDS

The SshCmCds contains the CDS with one or more credentials (i.e., username/password or public key entries) for authenticating inbound SSH sessions on the CM.

Type	Length	Value
98.3.1	n	

The definition of the CDS value is provided in Section 15.3.3.1.

The CM MUST configure the SSH server with the credentials in the SshCmCds TLV after properly authenticating the data via the signature/encryption or via the TLS download interface in Section 15.3.2, TLS-Based Authentication.

The CM MUST assume no SSH credentials are set when the SshCmCds TLV is not explicitly set.

A.1.2.3.2 SshCmCDSDownloadURL

This field indicates a URI where the CDS can be downloaded from.

Type	Length	Value
98.3.2	n	A URL to download the CDS from

When the SshCmCDSDownloadURL TLV is set, the CM MUST both download the CDS from the given URL and replace any existing credentials with the ones contained in the newly downloaded CDS (after performing any necessary decryption and/or signature validation as needed).

The CM MUST discard the downloaded CDS if that CDS does not contain any credentials that can be used by the CM to authenticate the CDS.

The CM MUST discard the downloaded CDS if the authentication/decryption of the CDS itself fails.

The CM MUST retain the existing credentials if the CDS download fails, or if decryption and/or signature validation fails.

A.1.3 Security Configuration Settings

Please refer to [DOCSIS MULPIv4.0] for additional configuration file settings that are grouped under TLV 99.

A.2 Parameter Guidelines

Below are recommended ranges and values for Baseline Privacy's various configuration and operational parameters. These ranges and default values may change as service providers gain operational experience running Baseline Privacy.

Table 57 - Recommended Operational Ranges for BPI+ Configuration Parameters

System	Name	Description	Minimum Value	Default Value	Maximum Value
CMTS	Authorization Lifetime	Lifetime, in seconds, CMTS assigns to new Authorization Key	1 day (86400 sec.)	7 days (604800 sec.	70 days (6048000 sec.)
CMTS	TEK Lifetime	Lifetime, in seconds, CMTS assigns to new TEK	30 min. (1800 sec.)	12 hours (43200 sec.)	7 days (604800 sec.)
CM	Authorize Wait Timeout	Auth Req retransmission interval from Auth Wait state	2 sec.	10 sec.	30 sec.
CM	Reauthorize Wait Timeout	Auth Req retransmission interval from Reauth Wait state	2 sec.	10 sec.	30 sec.
CM	Authorization Grace Time	Time prior to Authorization expiration CM begins reauthorization	5 min. (300 sec.)	10 min. (600 sec.)	35 days (3024000 sec).
CM	Operational Wait Timeout	Key Req retransmission interval from Op Wait state	1 sec.	10 sec.	10 sec.
CM	Rekey Wait Timeout	Key Req retransmission interval from Rekey Wait state	1 sec.	10 sec.	10 sec.
CM	TEK Grace Time	Time prior to newer TEK expiration CM begins re-keying	5 min. (300 sec)	1 hour (3600 sec.)	3.5 days (302399 sec)
CM	Authorize Reject Wait	Delay before re-sending Auth Request after receiving Auth Reject	10 sec.	60 sec.	10 min. (600 sec.)
CM	SA Map Wait Timeout	Map Request retransmission interval from Map Wait state	1 sec.	1 sec.	10 sec.
CM	SA Map Max Retries	Maximum number of times CM retries SA Map Request before giving up	0	4	10

The valid range (vs. recommended operational range) for Authorization and TEK lifetimes are:

• Authorization Lifetime Valid Range:	1 - 6048000 seconds
• TEK Lifetime Valid Range:	1 - 604800 seconds

The CMTS MUST support the valid range for Authorization and TEK lifetimes. The CM MUST support the valid range for Authorization and TEK lifetimes.

Annex B TFTP Options (Normative)

Network Working Group S. Zeng
Internet-Draft Cisco Systems, Inc.
Expires: December 1, 2006 D. R. Evans
ARRIS International, Inc.
May 30, 2006

Hardware and Network Address Options for TFTP
draft-evans-tftp-address-options-01.txt

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on December 1, 2006.

Copyright Notice

Abstract

The Hardware Address and Network Address options carry the hardware address and network address respectively of a client device that performs a Trivial File Transfer Protocol (TFTP) request.

1. Introduction

The Trivial File Transfer Protocol [2] (TFTP) is a simple protocol that allows a client to read a file from, or write a file to, a remote server.

In some networks, a proxy relays requests and responses between a TFTP client and a TFTP server. A router may also be present between the client and the server. In these cases, addressing information that identifies the client and that may be required by the server for authentication, file-generation or other purposes may not be readily available to the server. The options defined in this document allow the client or the proxy to provide the needed address(es) to the server.

An example of such a network would be one in which a large service provider deploys end-user devices that are provisioned using TFTP. The service provider might use the MAC address or the IP address of the end-user device in order to create a provisioning file for that particular device, or as keys in an internal database of end-user devices. However, if the TFTP request passes through a router, the MAC address of the end-user device is no longer available to the TFTP server. Similarly, if the request is proxied through an edge-of-network device the IP address of the end-user device can be unavailable to the TFTP server. By using the options defined in this document, the MAC and/or IP address(es) of the end-user device can be made available to the server.

The general mechanism used for adding options to TFTP messages is described in [4].

2. Use of TFTP

[6] discourages the use of TFTP, and cites several reasons for doing so. We similarly discourage use of the protocol. However, there are strictly limited scenarios in which it might be reasonable to deploy it. In particular, operators that support systems with a large deployed base and in which explicit steps have taken to address the security and other concerns of [6] may wish to continue to use TFTP. The options described in this document should be used only in such systems.

3. Terminology

The key words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL in this document are to be interpreted as described in [3].

4. Format of the Hardware Address option

The TFTP Read Request or Write Request packet is modified to include the hwaddr option. All named fields except "opc" are followed by a single-octet field containing the value zero.

```
+-------+--------+---+--------+---+--------+---+--------+---+
|  opc  |filename| 0 |  mode  | 0 | hwaddr | 0 |   ha   | 0 |
+-------+--------+---+--------+---+--------+---+--------+---+
```

opc The opcode field contains either a 1, for Read Requests, or 2, for Write Requests, as defined in [2].
filename The name of the file to be read or written, as defined in [2].
mode The mode of the file transfer: "netascii", "octet", or "mail", as defined in [2].
hwaddr The Hardware Address option, containing the case-insensitive string "hwaddr" in ASCII.
ha A hardware address. The format of hardware addresses is defined in Section 5.

5. Format of the Hardware Address

A hardware address comprises two comma-separated ASCII fields: hardware type and the address value.

hardware type A number representing the type of the hardware address. This document defines a single value, "1", representing an Ethernet address.
address value A representation of the hardware address. This document defines a single format, to be used in the case that the hardware type has the value "1". In this case, that address MUST be an Ethernet MAC address in the case-insensitive form "xx:xx:xx:xx:xx:xx".

6. Format of the Network Address option

The TFTP Read Request or Write Request packet is modified to include the netaddr option. All named fields except "opc" are followed by a single-octet field containing the value zero.

```
+-------+--------+---+--------+---+--------+---+--------+---+
|  opc  |filename| 0 |  mode  | 0 | netaddr| 0 |   na   | 0 |
+-------+--------+---+--------+---+--------+---+--------+---+
```

opc — The opcode field contains either a 1, for Read Requests, or 2, for Write Requests, as defined in [2].

filename — The name of the file to be read or written, as defined in [2].

mode — The mode of the file transfer: "netascii", "octet", or "mail", as defined in [2].

netaddr — The Network Address option, containing the case-insensitive string "netaddr" in ASCII.

na — A network address. The format of network addresses is defined in Section 7.

7. Format of the Network Address

A network address comprises two comma-separated ASCII fields: network type and the address value.

network type — A number representing the type of the network address. This document defines two values: "1" represents an IPv4 address; "2" represents an IPv6 address.

address value — A representation of the network address. This document defines two formats. If the network type has the value "1", the network address MUST be a dotted decimal IPv4 address as defined in [1]. If the network type has the value "2", the network address MUST be a case-insensitive IPv6 address in one of the formats specified by section 2.2 of [5].

8. Option Acknowledgement

[4] allows for the possibility that TFTP options will be acknowledged explicitly with an OACK packet. A TFTP server SHOULD NOT respond to the presence of a valid Hardware Address option or Network Address option by sending an OACK as defined in [4].

9. Errors

[4] allows for the possibility that TFTP options will contain errors.

For the options defined in this document, the server SHOULD return a TFTP ERROR message with ErrorCode value 8 if any of the following occurs:

1. An error when parsing an option;
2. An unknown hardware or network type;
3. An incorrectly formatted hardware or network address.

10. Security Considerations

TFTP provides no security safeguards; it relies on other layers to provide appropriate security where necessary. This document does not introduce any additional safeguards into TFTP. In the absence of other security measures, several possibilities exist for inappropriate behaviour:

- A client could populate the options defined in this document with incorrect but legal values. This could cause the TFTP server to behave in an undesirable manner (for example, it might report an incorrect hardware address to a backoffice system). This hazard can be circumvented by having a trusted device between the client and the server check and/or overwrite the values in the option fields.
- An attacker could replace correct option values with incorrect ones. This could cause the TFTP server to behave in an undesirable manner (for example, it might report an incorrect hardware address to a backoffice system).
- An attacker could insert legal but incorrect option values into a request that originally did not use the options defined in this document. This could cause the TFTP server to behave in an undesirable manner (for example, it might report an incorrect hardware address to a backoffice system).
- An attacker could return an ERROR message to the client even though there was no error in the request. This causes the requested transfer not to occur.
- An attacker could insert an option acknowledgement into a reply that did not originally contain that option. This results in undefined behaviour at the client.

Systems can take various steps to thwart these attacks. In general, TFTP should be used only on networks that provide either physical protection against attack or supports features such as client authentication and encryption of traffic between the client and server. The same features that allow TFTP to be used securely will generally thwart the above attacks. For example, all but the first attacks above are Man-in-the-Middle (MitM) attacks. These MitM attacks can be thwarted by properly encrypting the messages. Proper message encryption ensures that a potential attacker cannot perform any attacks that involve altering the values of the option fields.

11. IANA Considerations

This document has no actions for IANA.

12. References

12.1. Normative References

[1] Kirkpatrick, S., Stahl, M., and M. Recker, "Internet numbers", RFC 1166, July 1990.

[2] Sollins, K., "The TFTP Protocol (Revision 2)", STD 33, RFC 1350, July 1992.

[3] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[4] Malkin, G. and A. Harkin, "TFTP Option Extension", RFC 2347, May 1998.

[5] Hinden, R. and S. Deering, "Internet Protocol Version 6 (IPv6) Addressing Architecture", RFC 3513, April 2003.

12.2. Informative References

[6] Lear, E., "Uniform Resource Identifier (URI) Scheme and Applicability Statement for the Trivial File Transfer Protocol (TFTP)", RFC 3617, October 2003.

Authors' Addresses

Shengyou Zeng
Cisco Systems, Inc.
1414 Massachusetts Avenue
Boxborough, MA 01719
USA

Phone: +1 978.936.1609
Email: szeng@cisco.com

D. R. Evans
ARRIS International, Inc.
7912 Fairview Road
Boulder, CO 80303
USA

Phone: +1 303.494.0394
Email: N7DR@arrisi.com

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Disclaimer of Validity

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

Annex C DOCSIS 1.1/2.0 Dynamic Security Associations (Normative)

C.1 Introduction

Legacy CMs support the Dynamic Security Association functionality defined in this Annex when operating in DOCSIS 1.1 or 2.0 modes. DOCSIS 4.0 CMs do not support the Dynamic Security Association functionality defined in this Annex because they will never operate in DOCSIS 1.1 or 2.0 modes.

Dynamic Security Associations (Dynamic SAs) are SAs that a CMTS establishes and eliminates dynamically, in response to its enabling and disabling of downstream traffic flows that require DOCSIS security. These traffic flows may be initiated by the actions of:

- A CPE device attached to one of the CMTS's client CMs;
- An application server within the headend;
- An OSS;
- Other unspecified mechanisms.

Regardless of what triggers the establishment of a Dynamic SA within the CMTS, client CMs need a mechanism for learning the mapping of a particular secured downstream traffic flow to that flow's SAID. The SA Mapping state machine, defined in this section, defines how CMs query a CMTS for that mapping. This state machine controls the transmission of SA Map Request messages to a CMTS.

A CMTS can establish or eliminate Dynamic SAs in response to changes in IP group membership of downstream CPE devices. IGMP management can cause the CMTS to establish Dynamic SAs. If it detects IGMPv2 (see [RFC 3376]) join messages, the MDF-Disabled DOCSIS 3.0 CM triggers Map Request messages that query the CMTS for the mapping of the IP multicast group address contained in the IGMPv2 join message to an SA. If it detects IGMPv2 (see [RFC 3376]) join messages, the DOCSIS 1.1/2.0 CM triggers Map Request messages that query the CMTS for the mapping of the IP multicast group address contained in the IGMPv2 join message to an SA.

The SA mapping mechanism may map an IP multicast group to a Static SA or to a particular CM's Primary SA; thus, a CMTS's response to a mapping request may return any of the three types of SAs. The SA mapping mechanism, however, is the only mechanism by which a CM can learn the identity of Dynamic SAs.

C.2 Theory of Operation

Three BPKM messages support SA mappings: SA Map Request, SA Map Reply and SA Map Reject. A CM sends a Map Request to request the mapping of a known downstream flow to a SA. The Map Request carries attributes identifying the requesting CM and the downstream traffic flow whose SA mapping is being requested.

The CMTS MUST respond to a Map Request with either:

- A Map Reply, providing the CM with the requested SA mapping; or
- A Map Reject, signaling to the CM that either:
  - The CM is not authorized to receive the traffic flow identified in the Map Request; or
  - The requested traffic flow is not mapped to an SA.

If the CM does not receive either of these responses within a configurable retry timeout period, it re-sends the Map Request. If no response is received after a configurable maximum number of retries, the CM terminates the request.

If the CM receives a Map Reject, it ceases all further attempts to obtain the mapping. In the case where access to the downstream traffic flow is mapped to an SA and the requesting CM is not authorized access for that SA, the CM will be denied access because the CM cannot obtain keying material needed to decrypt the downstream traffic flows encrypted under that SA. In the case where the requested traffic flow is not encrypted (i.e., it is not mapped to an SA), the unencrypted traffic will simply be forwarded to the attached CPE device.

If the CM receives a Map Reply identifying the SA associated with the requested downstream traffic flow, the CM launches a TEK state machine for the SA, provided that:

- The CM is not already running a TEK state machine for that SA; and
- The CM supports the cryptographic suite and SAID identified in the Map Reply.

The CM may already be running a TEK state machine if the mapped SA is:

- A Dynamic SA mapped to another protected traffic flow to which the CM already has access;
- The requesting CM's Primary SA;
- A previously-learned Static SA.

The Map Reply includes an SA-Descriptor attribute that identifies both an SAID and the cryptographic suite employed by the SA. As with Static SAs, a CMTS MAY respond to a Map Request with an SA (either Static or Dynamic) that employs a cryptographic suite that the requesting CM does not support. The CM MUST NOT start TEK state machines for SAs whose cryptographic suites the CM does not support.

The TEK state machine controls the retrieval of the mapped SA's keying material.

Receipt of a Key Reject forces the termination of the TEK state machine.

There are two mechanisms for the CMTS to inform a client CM that it is not authorized to access a particular traffic flow:

1. Responding to a Map Request with a Map Reject, and
2. Responding to a Key Request with a Key Reject.

The CMTS SHOULD check a CM's authorization status prior to responding to a Map Request and, if the CM is not authorized to receive the traffic, respond with a Map Reject. By performing this check during the mapping exchange, a CM will be prevented from needlessly launching a TEK state machine and sending a Key Request for an SAID for which it is not authorized.

C.3 SA Mapping State Model

The SA Mapping state model specifies the mechanism by which a CM learns the mapping between a traffic flow and a Dynamic SA.

An SA Mapping state machine is started when an event, external to the SA Mapping State Model, triggers the need for a traffic-flow-to-SA mapping. This external event generates an internal {Map} event in the SA Mapping state machine.

The state machine is terminated if the CM receives no response after sending the maximum number of retries, or when the CM determines that it no longer requires the mapped SA's keying material. In the latter case, an external event sends an internal {Unmap} event to the SA Mapping state machine, forcing its termination. The CM MAY implement the {Unmap} event.

The SA Mapping state machine is presented as a state flow model (Figure 21) and as a state transition matrix (Table 58). The legacy CM use the state transition matrix and its associated text as the definitive specification of protocol actions associated with each state transition. A shaded cell within the state transition matrix implies that either the specific event should not occur within that state; if the event does occur, the legacy CM does not transition to another state.

When a CM requires access to a Dynamic SA's keying material, it establishes a TEK state machine for that Dynamic SA. While the Authorization state machine controls the establishment and termination of TEK state machines associated with Primary and Static SAIDs, it does not control the establishment and termination of TEK state machines associated with Dynamic SAs.

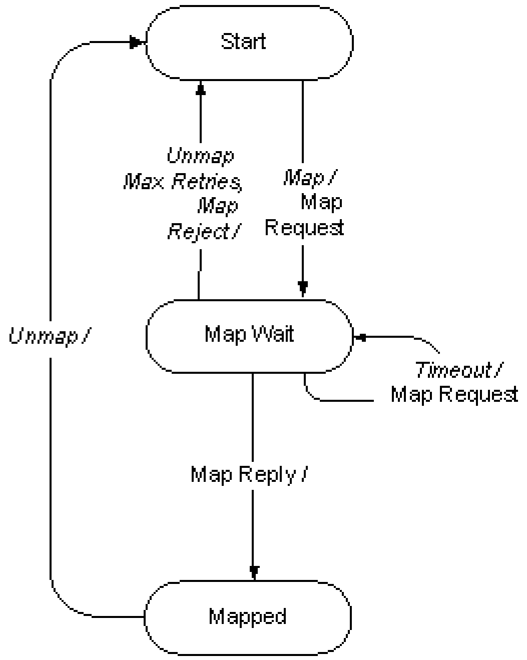

Figure 21 - SA Mapping State Machine Flow Diagram

Table 58 - Dynamic SAID State Transition Matrix

State Event or Rcvd Message	[Start]	[Map Wait]	[Mapped]
{Map}	[Map Wait]		
{Unmap}		[Start]	[Start]
{Map Reply}		[Mapped]	
{Map Reject}		[Start]	
{Timeout}		[Map Wait]	
{Max Retries}		[Start]	

Brief Description of States

[Start]

The initial state of the FSM.

[Map Wait]

The CM has sent the CMTS a Map Request and is waiting for a response.

[Mapped]

The CM has received a Map Reply and learned the requested SA mapping.

Brief Description of Messages

Map Request

Sent by CM to CMTS to request a SA mapping.

Map Reply

Positive CMTS response to a Map Request; contains the requested SA mapping.

Map Reject

Negative CMTS response to a Map Request; signals to the CM that either:

- The CM is not authorized access to the traffic flow identified in the Map Request; or
- The requested traffic flow is not mapped to an SA.

Brief Description of Events

{Map}

Triggers the start of the SA Mapping state machine. The {Map} event is linked to a CM event not defined by this specification.

{Unmap}

Terminates the SA Mapping state machine. The {Unmap} event is linked to a CM event not defined by this specification. The CM MAY implement the {Unmap} event.

{Map Reply}

CM has received an SA Map Reply message.

{Map Reject}

CM has received an SA Map Reject message.

{Timeout}

CM has timed out waiting for a response to an outstanding SA Map Request message.

{Max Retries}

CM has sent the maximum number of retries and not received a response.

Brief Description of Parameters

All configuration parameter values are obtained from the configuration file.

SA Map Wait Timeout

Timeout period between sending SA Map Request messages from SA Wait state (see Annex A).

SA Map Max Retries

This value specifies the maximum number of times that the CM may retry an SA Map Request.

Actions

Actions taken in association with state transitions are listed by <event/rcvd message> - <state> below:

[Start] + {Map} → [Map Wait]:

- Send SA Map Request
- Set Map Request retry timer to SA Map Wait Timeout
- Set Map Retry Count to 0

[Map Wait] + {Unmap} → [Start]:

- Clear Map Request retry timer
- Terminate SA Mapping state machine

[Mapped] + {Unmap} → [Start]:

- Terminate SA Mapping state machine

[Map Wait] + {Map Reply} → [Mapped]:

- Clear Map Request retry timer

[Map Wait] + {Map Reject} → [Start]:

- Clear Map Request retry timer
- Terminate SA Mapping state machine

[Map Wait] + {Timeout} → [Map Wait]:

- Send Map Request
- Set Map Request retry timer to SA Map Wait Timeout
- Increment Map Retry Count
- If Map Retry Count > SA Map Max Retries, generate Max Retries event

[Map Wait] + {Max Retries} → Start:

- Terminate SA Mapping state machine

Annex D Additions and Modifications for Chinese Specification (Normative)

This annex defines the Security requirements used in conjunction with the Chinese DOCSIS Architectures [C-DOCSIS].

This is an optional annex and in no way affects certification of equipment adhering to the North American technology option described in the sections referenced above.

The C-DOCSIS Cable Modem Termination System (CMTS) MUST support all the features and requirements as defined in this Security Specification. The C-DOCSIS Cable Modem (CM) MUST support all the features and requirements as defined in this Security Specification.

The following section identifies the main differences in security requirements as supported by devices deployed in C-DOCSIS architectures.

D.1 Security Requirement Differences for C-DOCSIS

- AES:
    - The CMTS MAY support Advanced Encryption Standard (AES) for traffic (packet PDU) encryption.

Appendix I Example Messages, Certificates, PDUs and Code File (Informative)

This appendix presents detailed examples that may be useful to implementers of the specification. The examples describe a typical key exchange: Authorization Info, Authorization Request, Authorization Reply, Key Request, and Key Reply. Details of the cryptographic calculations are provided at each step, and example certificates are included. The examples also include several Packet PDUs, encrypted using the keying material derived in the example key exchange.

This appendix is informative only. In the event of any discrepancy between this appendix and the main body of the specification or its associated annexes exists, the latter will take precedence.

I.1 Notation

In the examples here, packets are represented as a stream of octets, each octet in hexadecimal notation, sometimes with a text annotation. The order of transmission for the octets is left to right, top to bottom. For example, consider the following representation of a packet:

00 01 02 03	Description #1
04 05	
06 07 08	Description #2

The packet consists of 9 octets, represented in hexadecimal notation as "00", "01",..., "08." The octet represented by "00" is transmitted first, and the octet represented by "08" is transmitted last.

In the discussion of the examples, integer values are represented in either hexadecimal notation using an "0x" prefix or in decimal notation with no prefix. For example, the hexadecimal notation 0x12345 and the decimal notation 74565 represent the same integer value. All integer values are non-negative. Thus, 0xff represents the integer having value 255, not a negative value.

The BPKM protocol requires that devices generate and distribute DES keys without regard to parity. Devices ignore the value of the least significant bit of each octet. In the examples here, keys are represented without parity correction.

I.2 Authentication Info

The CM sends the following Authentication Info message:

0C 01 05 4F	Auth Info header (code=12,id=1,len=54f)
11 05 4C	CA Certificate header (type=17,len=0x54C)
30 82 05 B4 30 82 03 9C . . . 9D D1 DD 05 4B 03 AB 51	CA Certificate

The code field has value 0x0c, which identifies this as an Authentication Info message. The Length field has a value of 0x54F (1359), which is the number of octets that follow the Length field.

The only attribute is the CA Certificate. Details of the certificate are given below.

I.2.1 Device CA Certificate Details

The fields of the Device CA Certificate in the Authorization Info message are as follows:

30 82 05 48	Certificate header
30 82 03 30	TbsCertificate header
a0 03 02 01 02	Version, 0x2=v3
02 08 02 02 02 02 02 02 02 02	serial number, 0202020202020202
30 0D 06 09 2A 86 48 86 F7 0D 01 01 0B 05 00	Signature type sha256, null optional parameters
30 6E 31 0B	issuer header
30 09 06 03 55 04 06 13 02 55 53	countryName, US
31 12 30 10 06 03 55 04 0A 13 09 43 61 62 6C 65 4C 61 62 73	organizationName, CableLabs
31 12 30 10 06 03 55 04 0B 13 09 52 6F 74 20 43 41 30 31	organizationalUnitName, Root CA01
31 37 30 35 06 03 55 04 03 13 2E 45 78 61 6D 70 6C 65 20 43 61 62 6C 65 4C 61 62 73 20 52 6F 74 20 43 65 72 74 69 66 69 63 61 74 69 6F 6E 20 41 75 74 68 6F 72 69 74 79	commonName, Example CableLabs Root Certification Authority
30 1e	Validity header
17 0D 31 34 30 39 31 37 32 30 30 38 32 38 5A	notBefore, UTCTime 14/09/17 20:08:28Z
17 0D 34 39 30 39 31 37 32 30 30 38 32 38 5A	notAfter, UTCTime 49/09/17 20:08:28Z
30 72 31 0B	Subject header
30 09 06 03 55 04 06 13 02 55 53	countryName, US
31 12 30 10 06 03 55 04 0A 13 09 43 61 62 6C 65 4C 61 62 73	organizationName, CableLabs
31 14 30 12 06 03 55 04 0B 13 0B 44 65 76 69 63 65 20 43 41 30 31	organizationalUnitName, Device CA01
31 39 30 37 06 03 55 04 03 13 30 45 78 61 6D 70 6C 65 20 43 61 62 6C 65 4C 61 62 73 20 44 65 76 69 63 65 20 43 65 72 74 69 66 69 63 61 74 69 6F 6E 20 41 75 74 68 6F 72 69 74 79	commonName, Example CableLabs Device Certification Authority
30 82 01 A2	subject public key info header
30 0D 06 09 2A 86 48 86 F7 0D 01 01 01 05 00	public key algorithm type, RSA, null optional parameters
03 82 01 8F 00 30 82 01 8A 02 82 01 81	public key header
00 B6 A4 74 E7 A0 D4 BD 37 BB 5A E5 66 5A B8 5E 71 0B F8 71 5D 3F 9F 27 91 B0 41 6B FF 23 83 1A DB 16 48 12 9D 17 56 15 CE 7E DA 47 79 B8 0A 03 8A F7 50 A3 74 BE 83 68 19 AA A5 CE 0B 6C CA 17 6A FF 12 8A 8B EA E4 82 C7 5F 10 20 49 67 96 22 B1 2C 17 0E 93 02 99 B5 7A 29 44 4B D5 ED 9D F2 DA B1 16 A8 D3 99 C6 EE F5 7E 06 59 AF 15 D3 3D 8A 5D C5 08 25 17 7E 9C 53 E1 04 DF 93 C8 F5 37 1A BE C8 0C 8D EE BF 3A C6 D7 D3 B1 18 2F DC FB 20 F2 D6 CA 53 A5 EE 66 F0 54 B1 02 15 68 78 78	public key modulus (3072 bit)

26 C6 F7 CC B5 DF 89 9D A6 B8 93 E1 5D 7C FB 1A FD 52 2E E8 18 EE FF 07 8B B1 2E 51 C8 BC 72 ED 22 E9 32 4B 77 AC E8 7D A8 C5 79 81 91 9B 80 65 10 AC 41 46 42 1B F3 16 63 6A 82 EE A6 02 73 BA CB AA 50 20 A3 77 09 E4 B6 68 81 01 B3 5F B5 C4 B4 6E F0 B4 20 16 11 29 36 25 FA 35 B6 AD 4C 17 4F 8D EF CA 80 17 AA 21 77 6D DC 17 27 67 5B 6A 76 A6 C7 81 E7 73 BE E7 D8 A2 C7 B3 75 75 DE 5C 61 6F 69 C0 DC 6C 18 D1 F4 E8 77 20 4C 31 8F 21 CB 22 36 D3 F9 F8 47 20 D6 AD 5B E6 ED D8 EE CF 68 0E 18 BF C4 DF 12 E7 30 AB C7 3C 64 F8 77 FE 6D 74 E6 87 25 05 11 4A 35 E9 63 58 33 83 30 99 3A 1D 67 42 20 93 C2 DD 74 9C ED 9C E8 5D 47 9D 80 09 C5 E2 BC 6B 79 4E CE 17 F1 FE 55 0A 92 5B	
02 03 01 00 01	public key exponent header and value
A3 66 30 64	X.509 extensions header
30 0E 06 03 55 1D 0F 01 01 FF 04 04 03 02 01 06	Key usage, critical=true, CertSign, CRLSign
30 12 06 03 55 1D 13 01 01 FF 04 08 30 06 01 01 FF 02 01 00	Basic Constraints, critical=true, CA=true, pathlen:0
30 1D 06 03 55 1D 0E 04 16 04 14 42 A8 6E 6F 36 31 17 1B 17 EF 94 99 96 81 90 C9 B8 07 49 77	Subject key id
30 1F 06 03 55 1D 23 04 18 30 16 80 14 1F 43 E6 00 FC 16 2E 5A EA A5 4E 78 6F 04 64 E1 AE E6 A4 A8	Authority key id
30 0D 06 09 2A 86 48 86 F7 0D 01 01 0B 05 00	signature algorithm, sha256 with RSA encryption, null optional parameters
03 82 02 01	Signature header
00 6F 20 2F D3 C5 1F C2 41 5A 3E 85 89 E0 0B D8 EC 3B D7 B1 63 C3 4A 62 81 E4 BA 65 AE 76 43 62 A6 32 0D 72 1A 0A 5B 22 A4 F7 82 37 39 33 7E 99 DC CC D5 78 D9 C4 B5 A9 A3 E8 D9 38 83 17 C5 3B 14 CC DF 2B E1 E8 6C 62 8F A3 19 CB 23 CB B4 4A 7E A2 2E F3 98 00 71 17 C8 C2 B1 DE 12 9E B4 10 F2 40 85 B3 BB E6 0A 8D 93 41 B4 BE 51 94 45 3B BA 4B 7F C6 DF D8 94 C5 9F 82 B8 AC 6D 70 B9 65 66 59 7A F2 C1 CA CB 7A AF AA D4 7C 00 00 BB F5 EC 48 D6 E0 F0 FA 64 FC B6 4C 72 BB 40 56 96 E0 29 8F 98 2A FE E4 98 88 F9 D5 56 81 33 19 4E 13 43 A2 37 16 B0 EA 5B AC A4 9C 71 D8 7F DC FC 8A 46 16 62 9B 4D D1 77 F0 0B 7B AF 37 89 34 67 1C 65 DE DB 42 EF 26 36 55 13 17 A4 7D 1B BE 45 06 B4 ED 62 C8 DC 08 12 33 4D 71 99 06 98 6D FD DB 90 01 50 E8 CD A7 34 1B EB 19 82 D7 49 1F D1 31 44 C0 E3 B4 8B D3 32 95 92 4E 80 65 67 54 4D ED 61 1C 4C C0 29 92 03 9B 2E B3 F8 E6 86 63 EA F5 69 24 B4 BD 3F 59 E7 9C 82 B6 89 AE 84 F4 51 65 2D D6 68 49 58 44 DA 64 BD 77 FA A4 7E 22 4E B0 3E 2E 8B F1 E5 1E B9 1E 3B 70 E6 E6 43 09 04 DA 39 B2 C1 5F 12 74 18 46 CC 66 5B 03 1F 71 7B C3 19 22 87 5C 4C DA 0A D3 59 05 68 63 0E 0A 7A 13 2E A2 07 FB B8 8A 45 8F 5F 55 FC DD 11 CC D8 62 A0 55 7F 1F AD 83 8D 81 98 F2 6E E3 64 DE 52 37 80 41 0E 62 C9 AD D9 76 02 20 B8 A1 93 79 A4 41 4C B6 14 A5 0C B9 04 AA A3 BB 13 24 B7 35 C7 D6 F8 47 5B F1 7C 2B CE A5 1E 8C F6 E0 E5 9C 17 90 D2 A5 60 D7 98 47 8C 1B 90 63 2F C3 9A FA 16 A1 2A BA 78 0F 69 E1 7C 28 3F 41 4A E5 70 37 0A 7B 1D C5 C6 5B 09 5B 90 3D 63 75 A7 90 2C 5D 6B EA 68 B5 9F B0 CB 0A 6B B7 19 1B D3 C5 1A D8 E2 43	Signature value

Some of the fields in this example are identical in all CA certificates. These fields are:

- Version: v3
- Signature : SHA-256 with RSA, NULL parameters
- Public key algorithm type: RSA encryption, NULL parameters
- Public key exponent: 3-octet integer, value 0x10001
- Signature algorithm: SHA-256 with RSA, NULL optional encryption parameters

This is an example of a CableLabs Device CA certificate. While the countryName and organizationName are identical for both issuer and subject, the organizationalUnitName differs between issuer and subject.

Other fields that are example values include:

- The serial number: INTEGER of 8 octets, value 0x0202020202020202 (other CA certificates may use a different length)
- Not before: 2014/09/17 20:08:28 UTC
- Not after: 2049/09/17 20:08:28 UTC
- Public key modulus: INTEGER of 3072 bits, value 0x00B6A474... 550A925B with exponent of 0x10001
- X.509 v3 certificate extensions that include keyUsage, basicConstraints, subjectKeyIdentifier, and authorityKeyIdentifier
- Signature value: A BIT STRING representing the INTEGER value 0x6F202FD3... 1AD8E243. The signature is computed over the portion of the certificate that begins with the tbsCertificate header and ends with the public key exponent.)

I.3 Authorization Request

The CM sends the following Authorization Request:

`04 01 05 44`	`Auth Request header Code=4,Id=1,len=0x544`
`05 01 30`	`CM-Identification header, type=5`
`01 00 0e 39 33 33 38 33 57 56 47 2d 31 32 2d 35 34`	`Serial Number, type=1,len=e, v=93383WVG-12-54`
`02 00 03 00 10 18`	`Manufacturer ID, type=2, v=001018`
`03 00 06 00 10 18 de 02 3e`	`MAC Address`
`04 01 0F 30 82 01 0A 02 82 01 01`	`RSA Public Key Header`

`00 C1 32 23 2A 3B 2B 3F B1 0C A6 27 92 47 9C EE 19 B3 77 82`
`8D 04 DA 17 A1 A0 C5 5B 6F 4F 5E FE 11 1F CC 91 10 EB F9 12`
`C2 B1 6B 38 50 3B 23 BE E0 25 75 95 F0 77 1F 91 B0 18 15 DB`
`17 11 D3 A0 BA 91 EF 7E 00 9C 05 DE 0A 91 5F 3B 8F 6E 6D 57`
`6F 1D 3B 26 50 94 F5 E1 C3 59 F9 66 1C FB 79 6D C5 23 F9 9B`
`73 F7 B2 E0 D5 78 08 7D 25 FF 67 F9 AA 53 DD 26 4A 8D 1F 3B`
`9E AA 6C BB 70 E2 2A 51 78 62 B2 BB DB F7 B7 9A 34 67 F2 26`
`A2 AC BA 3E 00 E7 21 73 66 5B 7E BD 6A EF B1 A9 38 15 36 03`
`53 73 3A B4 30 94 8D 89 9A A1 0D 77 32 42 47 A1 CE 61 C4 2E`
`20 91 AE E4 43 5D BF 91 E3 91 3B 43 DF 6F 47 99 43 8B CF 10`
`C3 D7 D4 D6 3A E9 30 B7 87 70 14 FE 4D 5C A1 EF 1C F5 7F 9C`
`E7 3E C5 A5 0F 45 B4 D6 56 FD 2D 4D BD F2 E5 7B ED 0B C8 A8`
	`RSA Public Key modulus and exponent`

5E 42 F9 11 09 13 A8 CC D0 C4 57 A9 DF 54 3B A0 85 02 03 01 00 01	
12 03 fc	CM Certificate header
30 82 03 f8 30 82 02 60 . . . A8 B5 A7 D0 BB 3F 43 97	CM Certificate (see below for details)
13 00 0B	Security Capabilities header
15 00 04 01 00 03 00	Cryptographic Suite List
16 00 01 01	BPI Version
0c 00 02 00 00	SAID

The Code field has a value of 0x04, which identifies this as an Authorization Request packet. The Identifier field has value 0x01; this is an example value. The Length field has value 0x0544 (1348), which is the number of octets that follow the Length field.

The first attribute is the CM-Identification, which is a compound attribute consisting of the following sub-attributes: Serial Number, Manufacturer ID, MAC Address, and RSA Public Key. Example values are shown for these sub-attributes.

The Public Key is DER encoded and is similar to the example in [RSA2]. The modulus is a 2048-bit integer encoded in 0x101 (257) octets. In this example, the value of the encoded modulus is:

0x00C13223 ... 543BA085

0x00 is the most significant octet of the encoded modulus and 0x85 is the least significant. The exponent is a 3-octet integer with the value 0x010001.

The next attribute is the CM Certificate. Details of the certificate are given below.

NOTE: The MAC Address and RSA Public Key of the CM Identification match fields in the CM Certificate.

The next attribute is the Security Capabilities attribute, which is a compound attribute consisting of the Cryptographic Suite List and the BPI Version. In this example, the Cryptographic Suites listed is 56-bit DES with no authentication. The BPI Version is BPI+.

The final attribute is the CM's Primary SAID, whose value is equal to its Primary SID. In this example, the Primary SAID has value 0x0000.

I.3.1 CM Device Certificate Details

The fields of the CM Device Certificate in the Authorization Request message are as follows:

30 82 03 F8	certificate header
30 82 02 60	tbsCertificate header
a0 03 02 01 02	Version, v3=0x2
02 08 03 03 03 03 03 03 03 03	serial number, 0303030303030303
30 0D 06 09 2A 86 48 86 F7 0D 01 01 0B 05 00	Signature type, sha256 with RSA encryption, optional parameters null
30 72	issuer header
31 0B 30 09 06 03 55 04 06 13 02 55 53	countryName, US

31 12 30 10 06 03 55 04 0A 13 09 43 61 62 6C 65 4C 61 62 73	organizationName, CableLabs
31 14 30 12 06 03 55 04 0B 13 0B 44 65 76 69 63 65 20 43 41 30 31	organizationalUnitName, Device CA01
31 39 30 37 06 03 55 04 03 13 30 45 78 61 6D 70 6C 65 20 43 61 62 6C 65 4C 61 62 73 20 44 65 76 69 63 65 20 43 65 72 74 69 66 69 63 61 74 69 6F 6E 20 41 75 74 68 6F 72 69 74 79	commonName, Example CableLabs Device Certification Authority
30 1E	validity header
17 0D 31 34 30 39 31 37 32 30 31 39 35 37 5A	notBefore, 14/09/17 20:19:57Z
17 0D 33 34 30 39 31 37 32 30 31 39 35 37 5A	notAfter, 34/09/17 20:19:57Z
30 51	subject header
31 0B 30 09 06 03 55 04 06 13 02 55 53	countryName, US
31 11 30 0F 06 03 55 04 0A 13 08 42 72 6F 61 64 63 6F 6D	organizationName, Broadcom
31 13 30 11 06 03 55 04 0B 13 0A 44 75 6C 75 74 68 2C 20 47 41	organizational unit name, Duluth, GA
31 1A 30 18 06 03 55 04 03 13 11 30 30 3A 31 30 3A 31 38 3A 30 31 3A 30 32 3A 30 33	commonName (MAC address), 00:10:18:01:02:03
30 82 01 22	subject public key info header
30 0D 06 09 2A 86 48 86 F7 0D 01 01 01 05 00	public key algorithm type, RSA, null optional parameters
03 82 01 0F 00 30 82 01 0A 02 82 01 01	public key header
00 C1 32 23 2A 3B 2B 3F B1 0C A6 27 92 47 9C EE 19 B3 77 82 8D 04 DA 17 A1 A0 C5 5B 6F 4F 5E FE 11 1F CC 91 10 EB F9 12 C2 B1 6B 38 50 3B 23 BE E0 25 75 95 F0 77 1F 91 B0 18 15 DB 17 11 D3 A0 BA 91 EF 7E 00 9C 05 DE 0A 91 5F 3B 8F 6E 6D 57 6F 1D 3B 26 50 94 F5 E1 C3 59 F9 66 1C FB 79 6D C5 23 F9 9B 73 F7 B2 E0 D5 78 08 7D 25 FF 67 F9 AA 53 DD 26 4A 8D 1F 3B 9E AA 6C BB 70 E2 2A 51 78 62 B2 BB DB F7 B7 9A 34 67 F2 26 A2 AC BA 3E 00 E7 21 73 66 5B 7E BD 6A EF B1 A9 38 15 36 03 53 73 3A B4 30 94 8D 89 9A A1 0D 77 32 42 47 A1 CE 61 C4 2E 20 91 AE E4 43 5D BF 91 E3 91 3B 43 DF 6F 47 99 43 8B CF 10 C3 D7 D4 D6 3A E9 30 B7 87 70 14 FE 4D 5C A1 EF 1C F5 7F 9C E7 3E C5 A5 0F 45 B4 D6 56 FD 2D 4D BD F2 E5 7B ED 0B C8 A8 5E 42 F9 11 09 13 A8 CC D0 C4 57 A9 DF 54 3B A0 85	public key modulus
02 03 01 00 01	public key exponent
A3 33 30 31	X.509 extension header
30 0E 06 03 55 1D 0F 01 01 FF 04 04 03 02 05 A0	keyUsage, critical, DigitalSignature, key encipherment
30 1F 06 03 55 1D 23 04 18 30 16 80 14 42 A8 6E 6F 36 31 17 1B 17 EF 94 99 96 81 90 C9 B8 07 49 77	authorityKeyIdentifier, keyId
30 0D 06 09 2A 86 48 86 F7 0D 01 01 0B 05 00	signature algorithm, sha256 with RSA encryption, null optional parameters

03 82 01 81	signature header
00 43 E6 0C D1 80 A5 C5 FA D9 72 64 66 CA ED 21 B2 AF C6 83 9E B2 D8 79 1F 6C B3 02 1E C5 28 E1 33 68 E5 DC C0 D5 CF 98 1A FB 55 ED 88 29 67 BB B5 38 21 11 4A CE 68 30 75 F7 99 88 EB 99 C9 30 31 34 9D 06 0D DC BB 63 62 E5 C4 F6 BC AD 4D EA 28 4E C9 26 9B F8 1B F5 B5 63 F0 68 7E 82 2C D6 75 17 E6 EB 8E E8 1F 5A 1A A1 82 59 B5 04 A0 11 83 C0 C2 67 63 DF 68 49 C6 E7 DC A6 C0 EF 67 3E B9 6D FB B8 F8 E7 C4 3D FB 21 F7 17 F0 01 A0 B0 DE D5 D3 FF CD 2E 38 3C F4 7D 9D C2 B5 B5 50 7A 52 A9 6F 93 F8 3B 14 E1 28 AC 74 29 18 F4 81 1E A9 08 0A D8 9A 87 7D DB FD F6 E6 4E 06 05 33 95 28 61 A0 E1 AB 19 C4 96 C6 80 0C 15 DC C5 96 4B A2 0D 52 1D FD 78 0F D3 01 E7 D6 82 10 39 9C A8 A7 BA 48 EE 7C 46 64 85 79 1D 8E E2 01 21 37 9E BA AD 58 65 77 CE 6F E6 2D D6 19 AF 3F DB C1 0A 06 79 F3 42 A3 05 7B A2 6F 58 48 7D DD 1F B4 0B 79 15 2D 47 D8 CD 6F B9 D2 AA BF BB B8 A6 2F A5 31 F8 D1 04 10 45 50 F1 F1 45 75 F7 1D 4F 6C 7B 0A 5A 37 58 CF 7C 68 71 7E 3C 27 46 B7 A4 A5 AD E4 38 D2 3A 4D 54 08 3F 07 68 FE A4 35 12 87 39 C9 F0 5E 77 41 03 1A 2A 7A BD 61 B7 68 1B F6 BD 64 C1 E9 52 15 8F 8B 70 0A 34 FF 90 57 E1 FA 3A 83 4C 7A CC 0D E6 C5 AF 8D A8 B5 A7 D0 BB 3F 43 97	signature value

Some of the fields in this example are identical for all CM Certificates. These fields are:

- Version: v3
- Signature: SHA-256 with RSA, NULL parameters
- issuer first organizational unit name: Device CA01
- Public key algorithm type: RSA encryption, NULL optional encryption parameters
- Public key exponent: 3-octet integer, value 0x10001
- Signature algorithm: SHA-256 with RSA, NULL optional parameters

The issuer name of the CM certificate matches the subject name of the Device CA certificate. In this example, the matching issuer-name fields are:

- Country name: "US"
- Organization name: "CableLabs"
- First organizational unit name: "Device CA01"
- Common name: "Example CableLabs Device Certification Authority"

The other fields are example values. Some of these are:

- serial number: integer of 8 octets, value 0x0303030303030303 (other CM certificates may use a different length)
- Not before: 17/09/2014 20:19:57 UTC
- Not after: 17/09/2034 20:19:57 UTC
- Subject country name: "US"
- Subject organization name: "Broadcom"
- Subject organizational unit name: "Duluth, GA"
- Subject common name (MAC address): "00:10:18:01:02:03" (All CM certificates use a string of this length. The value matches the MAC Address attribute of the Authorization Request message.)

- Public key modulus: integer of length 2048 bits, encoded as 0x00C13223... 543BA085
- Signature value: BIT STRING representing the integer value 0x43E60CD1... BB3F4397 . The signature is computed over the portion of the certificate that begins with the tbsCertificate header and ends with the public key exponent, inclusive.

I.4 Authorization Reply

The CMTS sends the following Authorization Reply:

05 01 01 1F	Auth Reply header
07 01 00 28 27 55 DB F5 22 67 82 52 12 07 31 D2 B4 4D 35 8F 5F F6 51 54 7E A3 5C DD B9 11 B9 7F 92 1E B7 7E 47 77 B1 4A FE D2 AB 83 AD AE 29 8A 22 06 0A 76 F4 D7 C3 CB 80 45 1F 99 44 7E B9 50 90 F9 BD D4 B5 2E 25 D1 E6 4C A7 E9 45 0D B1 E9 63 45 38 F6 D9 83 79 82 3C 7B 93 80 8F DD 3D 0B 32 E3 4C 97 7D 8B 82 F0 16 17 FB F1 67 2E EE 39 57 39 B1 20 EB 13 B0 E1 D2 90 02 03 2C CA F9 3C A3 C1 6A CE 9A 5D BE ED F2 6E 86 2F 57 63 3B FE 13 DB D1 FE F0 1F 3E 9B D1 45 23 F6 F3 47 6F 87 61 01 01 57 C7 CE F3 D9 A1 14 05 DC 39 FA B8 BE E6 25 21 F4 FE BE 32 3D 6C 96 87 DE 17 AF F1 0A 83 35 19 93 C8 3D 89 BE 17 7B 36 CA 69 17 10 9B E1 19 24 F4 22 54 6E F3 04 38 1E B9 3C 5A 3B E0 6F 05 55 E4 85 C5 F9 81 13 9D E7 A9 8C 42 07 69 12 F8 2B 80 2D 6F 65 E9 4F 93 09 B3 C1 1E DD 89 68 95 20	Auth Key
09 00 04 00 09 3A 80	Key Lifetime
0A 00 01 07	Key Sequence number
17 00 0E	SA Descriptor header
0C 00 02 22 60	SAID
18 00 01 00	SA Type
14 00 02 01 00	Cryptographic Suite

The Code field has value 0x05, which identifies this as an Authorization Reply packet. The Identifier field has value 0x01, matching the Identifier field of the Authorization Request. The Length field has value 0x011F (287), which is the number of octets that follow the Length field.

The first attribute is the encrypted Authorization Key (attribute type 7). The attribute contains an authorization key which has been RSA-encrypted using the public key received by the CMTS from the CM in the Authorization Request message. The RSA-encrypted authorization key is an integer made up of 0x100 (256) octets. In this example, the value of the RSA-encrypted authorization key is 0xa2827 . . . 9520.

0x28 is the most significant octet of the RSA-encrypted authorization key and 0x20 is the least significant octet.

The second attribute is the Key Lifetime. In this example, the value is 0x00093a80 (604800) seconds, which is equivalent to 7 days.

The third attribute is the Key Sequence Number. In this example, the value is 0x07.

The remaining attributes are SA Descriptors. Each SA Descriptor is a compound attribute consisting of the following sub-attributes: SAID, SA Type, and Cryptographic Suite. In this example, a single SA Descriptor is included, corresponding to the SAID in the Authorization Request. The SA Type is Primary, and the Cryptographic Suite is 56-bit DES with no authentication.

I.4.1 Derivation of the Encryption Keys

The CM and CMTS each derive a key encryption key and two message authentication keys from the authorization key, using hashing. Details of the hashing calculations are given below. Here are the values of these keys for this example:

Authorization key	4e 85 27 ff c4 12 72 8e 61 84 de c9 20 b6 e0 64 f0 bc 0b 75
Key encryption key	76 b4 d4 2f 14 98 59 6a ab fe 72 94 15 7c 7d 62
Message authentication key, upstream	fe b9 f1 e2 46 a7 6d 7c a7 7b 5e b0 98 25 fd 0b 57 ca 90 c7
Message authentication key, downstream	93 d3 9d 70 c3 b6 f5 92 c4 6b d3 92 76 46 f4 f1 90 3a 52 fd

I.4.2 Encryption of the Authorization Key

The CMTS generates an authorization key of 20 octets. In this example, the value of the authorization key is:

```
4e 85 27 ff c4 12 72 8e 61 84 de c9 20 b6 e0 64 f0 bc 0b 75
```

The authorization key is encrypted using the RSAES-OAEP scheme in [RSA3]. The scheme makes use of a mask-generating function (MGF1), based on hashing using the SHA-1 hash as well as the OAEP padding scheme. The CMTS encrypts the authorization key using the public key received by the CMTS from the CM in the Authorization Request message. In this example, the encrypted value is as show below.

```
28 27 55 DB F5 22 67 82 52 12 07 31 D2 B4 4D 35 8F 5F F6 51 54 7E A3 5C DD B9 11 B9 7F
92 1E B7 7E 47 77 B1 4A FE D2 AB 83 AD AE 29 8A 22 06 0A 76 F4 D7 C3 CB 80 45 1F 99 44
7E B9 50 90 F9 BD D4 B5 2E 25 D1 E6 4C A7 E9 45 0D B1 E9 63 45 38 F6 D9 83 79 82 3C 7B
93 80 8F DD 3D 0B 32 E3 4C 97 7D 8B 82 F0 16 17 FB F1 67 2E EE 39 57 39 B1 20 EB 13 B0
E1 D2 90 02 03 2C CA F9 3C A3 C1 6A CE 9A 5D BE ED F2 6E 86 2F 57 63 3B FE 13 DB D1 FE
F0 1F 3E 9B D1 45 23 F6 F3 47 6F 87 61 01 01 57 C7 CE F3 D9 A1 14 05 DC 39 FA B8 BE E6
25 21 F4 FE BE 32 3D 6C 96 87 DE 17 AF F1 0A 83 35 19 93 C8 3D 89 BE 17 7B 36 CA 69 17
10 9B E1 19 24 F4 22 54 6E F3 04 38 1E B9 3C 5A 3B E0 6F 05 55 E4 85 C5 F9 81 13 9D E7
A9 8C 42 07 69 12 F8 2B 80 2D 6F 65 E9 4F 93 09 B3 C1 1E DD 89 68 95 20
```

Note the size of the encrypted authorization key is identical to the size of the modulus of the public key.

The CM decrypts the authorization key sent from the CMTS using its private key.

I.4.3 Hashing Details

The authorization key is hashed using the SHA-1 algorithm [FIPS 180-4] to produce the Key Encryption Key (KEK), the message authentication key for upstream, and the message authentication key for downstream.

The discussion here represents a hash calculation with a table that shows the input to the hash function and the resulting hash value. For reference, displayed below, is such a table that describes the example in [FIPS 180-4]:

Hash input	61 62 63 64 62 63 64 65 63 64 65 66 64 65 66 67 65 66 67 68 66 67 68 69 67 68 69 6a 68 69 6a 6b 69 6a 6b 6c 6a 6b 6c 6d 6b 6c 6d 6e 6c 6d 6e 6f 6d 6e 6f 70 6e 6f 70 71
Hash value	84 98 3e 44 1c 3b d2 6e ba ae 4a a1 f9 51 29 e5 e5 46 70 f1

I.4.3.1 KEK

The KEK is computed using the following hash calculation:

Hash input	53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 53 4e 85 27 ff c4 12 72 8e 61 84 de c9 20 b6 e0 64 f0 bc 0b 75
Hash value	76 b4 d4 2f 14 98 59 6a ab fe 72 94 15 7c 7d 62 b0 df e6 3b

The input is the octet 0x53, repeated 63 times, followed by the 20 octets of the authorization key. The order in which the octets of the authorization key are digested is the same as the order in which they appear in the EM encryption block.

The hash value is 20 bytes long. The first 16 bytes are the KEK.

I.4.3.2 Message Authentication Keys

The upstream message authentication key is computed using the following hash calculation:

Hash input	5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 5c 4e 85 27 ff c4 12 72 8e 61 84 de c9 20 b6 e0 64 f0 bc 0b 75
Hash value	fe b9 f1 e2 46 a7 6d 7c a7 7b 5e b0 98 25 fd 0b 57 ca 90 c7

The input is the octet 0x5c, repeated 63 times, followed by the 20 octets of the authorization key. The order in which the octets of the authorization key are digested is the same as in the KEK calculation.

The hash value is 20 octets long. The 20 octets make up the upstream message authentication key.

The downstream message authentication key is computed using the following hash calculation:

Hash input	3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 3a 4e 85 27 ff c4 12 72 8e 61 84 de c9 20 b6 e0 64 f0 bc 0b 75
Hash value	93 d3 9d 70 c3 b6 f5 92 c4 6b d3 92 76 46 f4 f1 90 3a 52 fd

This is similar to the computation for the upstream case, except that value 0x3a replaces value 0x5c.

I.4.3.3 Mask- Generation Function

The mask-generation function (MGF) is constructed from SHA-1 hash operations. Each hash operation generates 20 octets of mask data. The number of hash operations performed depends on the size of the mask that is needed.

Quantity SEED_MASK is formed by applying the MGF to MASKED_DB. Since SEED_MASK is 20 octets long, this requires only one hash operation:

Hash input	04 29 6a b7 1f a2 a1 7f 96 60 d7 96 47 33 9d 2d bc a3 a1 32 37 ac 86 06 7c b5 ec 97 d2 d0 9e 01 30 2b 10 91 3a ec 3f d9 a1 2f c4 e9 8d 18 88 95 f6 9c ea 17 23 9f 5d d5 f1 4d 25 8e 9e 6d 7d 3c ca 55 fe 0e ee 2d 0d 7e 5b 64 b6 79 44 76 cc 3f 6e ac 99 3a ae 14 3f d4 0b f8 c3 f2 6b 2a 3c 9b 97 ac 91 6c 7c e4 c5 5f 7b cf 17 00 00 00 00
Hash value	b4 b6 f1 bf a6 b3 a1 7e 95 82 d3 b8 93 71 b6 7f 45 31 9e 82

The input data to the hash operation are the 107 octets MASKED_DB followed by four octets of value 0. The output of the hash operation is the value of SEED_MASK.

Quantity DB_MASK is formed by applying the MGF to SEED. Since DB_MASK is 107 octets long, this requires six hash operations:

Hash input	ad 9c af 8d f8 26 fe af b5 df fd 95 de 7e 97 cc e9 4b 6d 00 00 00 00
Hash value	de 10 c9 59 41 c9 ea 72 a4 35 68 79 d2 53 85 bd 13 7b a6 3b
Hash input	ad 9c af 8d f8 26 fe af b5 df fd 95 de 7e 97 cc e9 4b 6d 00 00 00 01
Hash value	37 ac 86 06 7c b5 ec 97 d2 d0 9e 01 30 2b 10 91 3a ec 3f d9

Hash input	ad 9c af 8d f8 26 fe af b5 df fd 95 de 7e 97 cc e9 4b 6d 00 00 00 02
Hash value	a1 2f c4 e9 8d 18 88 95 f6 9c ea 17 23 9f 5d d5 f1 4d 25 8e

Hash input	ad 9c af 8d f8 26 fe af b5 df fd 95 de 7e 97 cc e9 4b 6d 00 00 00 03
Hash value	9e 6d 7d 3c ca 55 fe 0e ee 2d 0d 7e 5b 64 b6 79 44 76 cc 3f

Hash input	ad 9c af 8d f8 26 fe af b5 df fd 95 de 7e 97 cc e9 4b 6d 00 00 00 04
Hash value	6e ac 99 3a ae 14 3e 9a 8e df 3c 36 79 58 b2 fa 13 72 58 4c

Hash input	ad 9c af 8d f8 26 fe af b5 df fd 95 de 7e 97 cc e9 4b 6d 00 00 00 05
Hash value	ca 04 a1 af c7 c4 62 3a df 6f 33 ec e2 cd 2c 7f b7 7e 48 19

The input data to each hash operation are the 20 octets of SEED followed by a four-octet value. The four-octet value counts the integer values 0, 1, 2, 3, 4, 5 on successive hash operations. The outputs of the six hash operations are concatenated into a 120-octet result, and the first 107 octets of the result constitute DB_MASK.

I.5 Key Request

The CM sends the following Key Request:

07 73 00 d0	Key Request Header
05 00 ad	CM-Identification header
01 00 0c 30 30 30 30 30 30 31 32 33 34 35 36	Serial Number
02 00 03 25 53 41	Manufacturer ID
03 00 06 00 00 ca 01 04 01	MAC Address
04 00 8c 30 81 89 02 81 81 00 e0 6c 8d be b2 8b c9 f3 a6 3d a1 12 ea f7 99 f7 3d 3e fa a3 b1 e2 42 95 71 b5 71 d2 32 7a da 10 40 e2 5b 09 74 69 08 78 46 37 71 34 3e 69 a7 37 6d f8 70 1d aa a5 34 b0 33 a3 43 ac 4d eb 41 5e 0a 8a fd a6 0a 4b 09 7f 5a 18 f2 9e c2 22 a6 6b 9a 69 73 22 d5 37 c9 63 b0 88 f5 60 5d 99 16 33 54 53 30 ed 35 de 0c 87 3b 54 ba 59 22 3e b2 79 90 96 61 db f3 4a 37 18 4c 7f a8 ca ee d6 31 02 03 01 00 01	RSA public key
0a 00 01 07	Key Sequence Number
0c 00 02 22 60	SAID
0b 00 14 86 b8 33 b7 48 9c 4b a1 51 67 44 d7 a6 e6 ca 21 33 f5 22 9e	HMAC digest

The code field has value 0x07, which identifies this as a Key Request packet. The identifier field has a value of 0x73, which is an example value, obtained by incrementing the Identifier value in the Authorization Request. The Length field has value 0xk00d0 (208), which is the number of octets that follow the Length field.

The first attribute is the CM Identification. This is a compound attribute, identical to that in the Authorization Request.

The second attribute is the Key Sequence Number, which identifies the authorization key. The value is identical to that in the Authorization Reply.

The third attribute is the SAID for which a key is being requested. This SAID value was contained in the Authorization Reply.

The final attribute is the HMAC Digest. The digest consists of 20 octets. It is computed using the upstream message authentication key. The digest is calculated over all octets of the Key Request packet, excluding the 23 octets of the HMAC Digest attribute itself. Details of the digest calculation are given below.

I.5.1 HMAC Digest Details

The HMAC digest is computed using the HMAC authentication method defined in [RFC 2104], with SHA-1 as the hash function. Example calculations of HMAC using SHA-1 are presented in [RFC 2202].

The discussion here represents an HMAC calculation using a table that shows the key, the input to the HMAC function, and the resulting HMAC digest. For reference, here is a table that describes test case #2 of the HMAC-SHA-1 examples in [RFC 2202]:

Key	4a 65 66 65
HMAC input	77 68 61 74 20 64 6f 20 79 61 20 77 61 6e 74 20 66 6f 72 20 6e 6f 74 68 69 6e 67 3f
HMAC digest	ef fc df 6a e5 eb 2f a2 d2 74 16 d5 f1 84 df 9c 25 9a 7c 79

The HMAC digest of the Key Request packet is computed using the following HMAC calculation:

Key	fe b9 f1 e2 46 a7 6d 7c a7 7b 5e b0 98 25 fd 0b 57 ca 90 c7
HMAC input	07 73 00 d0 05 00 ad 01 00 0c 30 30 30 30 30 30 31 32 33 34 35 36 02 00 03 25 53 41 03 00 06 00 00 ca 01 04 01 04 00 8c 30 81 89 02 81 81 00 e0 e0 6c 8d be b2 8b c9 f3 a6 3d a1 12 ea f7 99 f7 3d 3e fa a3 b1 e2 42 95 71 b5 71 d2 32 7a da 10 40 e2 5b 09 74 69 08 78 46 37 71 34 3e 69 a7 37 6d f8 70 1d aa a5 34 b0 33 a3 43 ac 4d eb 41 5e 0a 8a fd a6 0a 4b 09 7f 5a 18 f2 9e c2 22 a6 6b 9a 69 73 22 d5 37 c9 63 b0 88 f5 60 5d 99 16 33 54 53 30 ed 35 de 0c 87 3b 54 ba 59 22 3e b2 79 90 96 61 db f3 4a 37 18 4c 7f a8 ca ee d6 31 02 03 01 00 01 0a 00 01 07 0c 00 02 22 60
HMAC digest	86 b8 33 b7 48 9c 4b a1 51 67 44 d7 a6 e6 ca 21 33 f5 22 9e

The key is the upstream message authentication key. The input consists of all octets of the Key Request packet, excluding the HMAC Digest attribute. The octets of the digest are the contents of the HMAC Digest attribute.

I.6 Key Reply

The CMTS sends the following Key Reply:

08 73 00 68	Key Reply header
0a 00 01 07	Key Sequence Number (authorization key)
0c 00 02 22 60	SAID
0d 00 21	TEK Parameters header
08 00 08 b6 4d 54 8c 3f 6b 25 69	TEK Key
09 00 04 00 00 a8 c0	Key Lifetime
0a 00 01 02	Key Sequence Number (TEK)
0f 00 08 81 0e 52 8e 1c 5f da 1a	DES CBC IV
0d 00 21	TEK Parameters header
08 00 08 5e bd 03 aa 5e d5 e2 94	TEK Key
09 00 04 00 01 51 80	Key Lifetime
0a 00 01 03	Key Sequence Number (TEK)
0f 00 08 25 35 67 c3 09 21 8c 2c	DES CBC IV
0b 00 14 a5 e3 33 25 ea 72 f8 50 1c 2a b6 65 45 6b cc de 8b 4f 22 02	HMAC Digest

The Code field has value 0x08, which identifies this as a Key Reply packet. The Identifier has 0x73, matching the value in the Key Request. The Length field has value 0x68 (104), which is the number of octets that follow the Length field.

The Key Sequence Number attribute identifies the authorization key. It matches the value in the Key Request.

The SAID attribute identifies the SAID for with a TEK is being supplied. It matches the value in the Key Request.

Two TEK Parameters attributes are included, the first for the older generation of key parameters and the second for the newer. Each TEK Parameters attribute is a compound attribute consisting of the following sub-attributes: TEK Key, Key Lifetime, Key Sequence Number, and DES CBC IV.

The TEK Key consists of 8 octets. It contains the TEK, encrypted using triple-DES-ECB with the KEK derived from the authorization key. Details of the triple-DES-ECB calculation are given below.

The Key Lifetime sub-attribute refers to the TEK. In this example, the value for the older TEK is 0x0000a8c0 (43200) seconds, equivalent to 12 hours, and the value for the newer TEK is 0x00015180 (86400) seconds, equivalent to 24 hours.

The Key Sequence Number sub-attribute identifies the TEK. In this example, the value for the older TEK is 0x02, and the value for the newer TEK is 0x03.

The DES CBC IV sub-attribute consists of 8 octets. It specifies the Initialization Vector to be used with the TEK.

The final attribute is the HMAC Digest. It consists of 20 octets. It is computed in a manner similar to that in the Key Reply, except that the downstream message authentication key is used instead of the upstream key. Details of the HMAC calculation are given below.

After the CM processes the Key Reply packet, the CM and CMTS each share two generations of TEK and IV. Here are the values of these parameters for this example:

Older TEK	e6 60 0f d8 85 2e f5 ab
Older IV	81 0e 52 8e 1c 5f da 1a
Newer TEK	b1 d7 4f c9 64 68 f7 58
Newer IV	25 35 67 c3 09 21 8c 2c

I.6.1 TEK Encryption Details

The CMTS generates a TEK of 8 octets. In this example, the value of the TEK is:

`e6 60 0f d8 85 2e f5 ab.`

This is the first TEK of the Key Reply message.

The TEK is encrypted using triple-DES-ECB encryption. The encryption key is the KEK:

`76 b4 d4 2f 14 98 59 6a ab fe 72 94 15 7c 7d 62.`

Triple-DES-ECB encryption is described here in terms of several iterations of DES-ECB encryption or decryption. DES-ECB is defined in [FIPS 46-3].

The discussion here represents a DES-ECB encryption or decryption operation using a table that shows the key, the input, and the output. For reference, here are tables that describe the example in Table B1 of [FIPS 46-3]:

Mode	ECB encryption
Key	01 23 45 67 89 ab cd ef
DES input	4e 6f 77 20 69 73 20 74
DES output	3f a4 0e 8a 98 4d 48 15

Mode	ECB decryption
Key	01 23 45 67 89 ab cd ef
DES input	3f a4 0e 8a 98 4d 48 15
DES output	4e 6f 77 20 69 73 20 74

NOTE: [FIPS 46-3] calls for the least significant bit of each octet in the key to be adjusted so that the octet has odd parity. This is evident in the key in the above example. The BPKM protocol does not require odd parity. BPKM generates and distributes 8-octet DES keys of arbitrary parity, and it requires that implementations ignore the value of the least significant bit of each octet.

The TEK is triple-DES-ECB encrypted using the following three DES-ECB operations:

Mode	ECB encryption
Key	76 b4 d4 2f 14 98 59 6a
DES input	e6 60 0f d8 85 2e f5 ab
DES output	c3 94 31 f5 8d f9 1d bf

Mode	ECB decryption
Key	ab fe 72 94 15 7c 7d 62
DES input	c3 94 31 f5 8d f9 1d bf
DES output	44 b0 94 4e ab 04 4c 23

Mode	ECB encryption
Key	76 b4 d4 2f 14 98 59 6a
DES input	44 b0 94 4e ab 04 4c 23
DES output	b6 4d 54 8c 3f 6b 25 69

The first and third operations are DES-ECB encryption; the key for each is the first eight octets of the KEK. The second operation is DES-ECB decryption; the key is the last eight octets of the KEK. The input to the first operation is the TEK to be encrypted. The input to the second operation is the output of the first, and the input to the third operation is the output of the second. The output of the third operation is the encrypted TEK; this is conveyed in the TEK Key sub-attribute of the Key Reply message.

I.6.2 HMAC Details

The HMAC digest of the Key Reply packet is computed by a method similar to that of the Key Request packet. The key is the downstream message authentication key. Here are the details of the HMAC calculation:

Key	93 d3 9d 70 c3 b6 f5 92 c4 6b d3 92 76 46 f4 f1 90 3a 52 fd
HMAC input	08 73 00 68 0a 00 01 07 0c 00 02 22 60 0d 00 21 08 00 08 b6 4d 54 8c 3f 6b 25 69 09 00 04 00 00 a8 c0 0a 00 01 02 0f 00 08 81 0e 52 8e 1c 5f da 1a 0d 00 21 08 00 08 5e bd 03 aa 5e d5 e2 94 09 00 04 00 01 51 80 0a 00 01 03 0f 00 08 25 35 67 c3 09 21 8c 2c
HMAC digest	a5 e3 33 25 ea 72 f8 50 1c 2a b6 65 45 6b cc de 8b 4f 22 02

I.7 Packet PDU Encryption (DES)

The first 12 octets of the Packet PDU, containing the Ethernet/802.3 destination and source addresses (DA/SA), are not encrypted. The remaining octets of the Packet PDU are encrypted in this example using DES-CBC mode with special handling of residual termination blocks that are less than 64 bits. The combination of DES-CBC and residual block processing ensures that the encryption does not change the length of the packet. The encryption key is the TEK corresponding to the key sequence number of the packet's Privacy Extended Header.

This specification describes the residual block processing as follows:

> The next-to-last ciphertext block is encrypted a second time, using the ECB mode of the encryption algorithm, and the least significant n bits of the result are XORed with the final n bits of the payload to generate the short final cipher block. In order for the receiver to decrypt the short final cipher block, the receiver encrypts the next-to-last ciphertext block using the ECB mode of the encryption algorithm, and XORs the left-most n bits with the short final cipher block in order to recover the short final cleartext block.

An alternative description of this procedure, which is equivalent to the description above, is as follows:

> Given a final block having n bits, where n is less than the length of the defined block for the cipher, the n bits are padded up to a block of the correct length by appending bits of arbitrary value to the right of the n payload bits. The resulting block is encrypted using the CFB<n> mode (where <n> is the length, in bits, of the block for the cipher in question) with the next-to-last ciphertext block serving as initialization vector for the CFB<n> operation. The leftmost n bits of the resulting ciphertext are used as the short cipher block. In the special case where the PDU is less than the length of the block for the cipher, the procedure is the same as for a short final block, with the provided initialization vector serving as the initialization vector for the CFB<n> operation.

The alternative description produces the same ciphertext as does the description in the body of this specification. In the alternative description, however, no mention is made of combining ECB encryption with XORs. These operations are internal to CFB, just as they are internal to CBC. The alternative description is convenient here because it allows residual block processing to be illustrated using CFB examples in [FIPS 46-3].

The Packet PDU includes the DA, SA, and Type/Len fields. In the examples here, no effort is made to use correct values for these fields. As a result, the examples here are not valid packets suitable for transmission. The intent of the examples is to illustrate encryption details only.

In these examples, the TEK and IV are taken from the example Key Reply packet described above.

I.7.1 CBC Only

When the number of octets to be encrypted is a multiple of 8, the encryption mode is DES-CBC as defined in [FIPS 46-3]. The encryption key and IV are as conveyed in the Key Reply packet.

The discussion here represents a DES-CBC encryption using a table that shows the key, IV, plaintext input, and ciphertext output. For reference, here is a table that describes the example in Table C1 of [FIPS 46-3]:

Mode	CBC
Key	01 23 45 67 89 ab cd ef
IV	12 34 56 78 90 ab cd ef
Plaintext	4e 6f 77 20 69 73 20 74 68 65 20 74 69 6d 65 20
Ciphertext	e5 c7 cd de 87 2b f2 7c 43 e9 34 00 8c 38 9c 0f

Suppose that the PDU, prior to encryption, is as follows:

DA	01 02 03 04 05 06
SA	f1 f2 f3 f4 f5 f6
Type/Len	00 01
User Data	02 03 04 05 06 07 08 09 0a 0b
CRC	88 41 65 06

The DES-CBC encryption is performed as follows:

Mode	CBC
Key	e6 60 0f d8 85 2e f5 ab
IV	81 0e 52 8e 1c 5f da 1a
Plaintext	00 01 02 03 04 05 06 07 08 09 0a 0b 88 41 65 06
Ciphertext	0d da 5a cb d0 5e 55 67 9f 04 d1 b6 41 3d 4e ed

The PDU, after encryption, looks like this:

DA	01 02 03 04 05 06
SA	f1 f2 f3 f4 f5 f6
Type/Len	0d da
User Data	5a cb d0 5e 55 67 9f 04 d1 b6
CRC	41 3d 4e ed

I.7.2 CBC with Residual Block Processing

When the number of octets to be encrypted is greater than 8 and is not a multiple of 8, the encryption mode is a combination of DES-CBC and DES-CFB64.

Encryption begins in DES-CBC mode. DES-CBC is used to process as many complete DES blocks as are present. The encryption key and IV are as conveyed in the Key Reply packet.

After the DES-CBC encryption, there is some number of octets that have not been encrypted. These octets are encrypted using DES-CFB64 mode. DES-CFB64 is the "64-bit Cipher Feedback Mode" defined in [FIPS 46-3]. The encryption key is as in the Key Reply packet. The IV is the last 8 octets of ciphertext produced by the DES-CBC processing.

The example here represents a DES-CFB64 encryption using a table that shows the key, IV, plaintext input, and ciphertext output. For reference, here is a table that describes the example in Table D3 of [FIPS 46-3]:

Mode	CFB64
Key	01 23 45 67 89 ab cd ef
IV	12 34 56 78 90 ab cd ef
Plaintext	4e 6f 77 20 69 73 20 74 68 65 20 74 69 6d 65 20
Ciphertext	f3 09 62 49 c7 f4 6e 51 a6 9e 83 9b 1a 92 f7 84

Suppose that the PDU, prior to encryption, is as follows:

DA	01 02 03 04 05 06
SA	f1 f2 f3 f4 f5 f6
Type/Len	00 01
User Data	02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e
CRC	91 d2 d1 9f

The total number of octets to be encrypted is 19. The first 16 octets are processed using DES-CBC encryption, and the last 3 octets using DES-CFB64 encryption.

The DES-CBC encryption is performed as follows:

Mode	CBC
Key	e6 60 0f d8 85 2e f5 ab
IV	81 0e 52 8e 1c 5f da 1a
Plaintext	00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 91
Ciphertext	0d da 5a cb d0 5e 55 67 51 47 46 86 8a 71 e5 77

The DES-CFB64 encryption is performed as follows:

Mode	CFB64
Key	e6 60 0f d8 85 2e f5 ab
IV	51 47 46 86 8a 71 e5 77
Plaintext	d2 d1 9f 00 00 00 00 00
Ciphertext	ef ac 88 e8 ee 80 33 14

The key is the same as used for the DES-CBC encryption operation. The IV is the last 8 octets of ciphertext generated by the DES-CBC operation.

Notice that 5 octets of value 0 have been appended to the 3 plaintext octets. The values of these appended plaintext octets have no effect on the values of the first 3 ciphertext octets, which are the only ciphertext octets of interest. Arbitrary values can be used for the appended plaintext octets.

The PDU, after encryption, looks like this:

DA	01 02 03 04 05 06
SA	f1 f2 f3 f4 f5 f6
Type/Len	0d da
User Data	5a cb d0 5e 55 67 51 47 46 86 8a 71 e5
CRC	77 ef ac 88

I.7.3 Runt Frame

When the number of octets to be encrypted is less than 8, the encryption mode is DES-CFB64. The encryption key and IV are as conveyed in the Key Reply packet.

Suppose that the PDU, prior to encryption, is as follows:

DA	01 02 03 04 05 06
SA	f1 f2 f3 f4 f5 f6
Type/Len	00 01
User Data	02
CRC	88 ee 59 7e

The DES-CFB64 encryption is performed as follows:

Mode	CFB64
Key	e6 60 0f d8 85 2e f5 ab
IV	81 0e 52 8e 1c 5f da 1a
Plaintext	00 01 02 88 ee 59 7e 00
Ciphertext	17 86 a8 03 a0 85 75 01

An octet of value 0 has been appended to the 7 plaintext octets. The value of this appended plaintext octet has no effect on the values of the first 7 ciphertext octets, which are the only ciphertext octets of interest. An arbitrary value can be used for the appended plaintext octet.

The PDU, after encryption, looks like this:

DA	01 02 03 04 05 06
SA	f1 f2 f3 f4 f5 f6
Type/Len	17 86
User Data	a8
CRC	03 a0 85 75

I.7.4 40-bit Key

The BPKM protocol always generates and distributes 56-bit DES keys. When 40-bit encryption is required, the 56-bit DES key is converted within an implementation to a 40-bit key by masking off (to zero) 16 of the 56 bits of a TEK.

A TEK has 8 octets, each octet containing 7 bits of key and 1 parity bit. Here is the procedure for converting a TEK to a 40-bit key:

- the first two octets of the TEK are set to 0;
- the two most significant bits of the third octet of the TEK are set to 0;
- the remaining five octets of the TEK are unchanged.

For example, if the TEK distributed by the BPKM protocol is:

`ff ff ff ff ff ff ff ff,`

then the conversion to 40 bits yields the TEK

`00 00 3f ff ff ff ff ff.`

Except for this conversion of the TEK value, the procedure for 40-bit encryption of a Packet PDU is identical to the case of 40-bit encryption.

To illustrate 40-bit encryption, a previous example of Packet PDU is repeated here, with the TEK converted to 40 bits.

Suppose that the Packet PDU, prior to encryption, is as follows:

DA	01 02 03 04 05 06
SA	f1 f2 f3 f4 f5 f6
Type/Len	00 01
User Data	02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e
CRC	91 d2 d1 9f

The total number of octets to be encrypted is 19. The first 16 octets are processed using DES-CBC encryption, and the last 3 octets using DES-CFB64 encryption.

The DES-CBC encryption is performed as follows:

Mode	CBC
Key	00 00 0f d8 85 2e f5 ab
IV	81 0e 52 8e 1c 5f da 1a
Plaintext	00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 91
Ciphertext	44 c8 4a 41 14 67 56 a2 dc 64 8f b0 dc 1e 1e 86

The key is the TEK conveyed in the Key Reply message, converted to a 40-bit key. The IV is as conveyed in the Key Reply message.

The DES-CFB64 encryption is performed as follows:

Mode	CFB64
Key	00 00 0f d8 85 2e f5 ab
IV	dc 64 8f b0 dc 1e 1e 86
Plaintext	d2 d1 9f 00 00 00 00 00
Ciphertext	f1 42 aa a3 e4 9b eb 29

The key is the same as used for the DES-CBC encryption operation. The IV is the last 8 octets of ciphertext generated by the DES-CBC operation.

The Packet PDU, after encryption, looks like this:

DA	01 02 03 04 05 06
SA	f1 f2 f3 f4 f5 f6
Type/Len	44 c8
User Data	4a 41 14 67 56 a2 dc 64 8f b0 dc 1e 1e
CRC	86 f1 42 aa

I.8 Encryption of PDU with Payload Header Suppression (DES)

These examples show how encryption is applied to a PDU when Payload Header Suppression (PHS) is applied. The examples use an RTP [RFC 3550] Voice over IP payload. In the examples, no effort is made to use correct values for the fields of the PDU. As a result, the examples here are not valid packets suitable for transmission. The intent of the examples is to illustrate encryption details only.

I.8.1 Downstream

Suppose that the PDU, after PHS and prior to encryption, is as follows:

DA	01 02 03 04 05 06
SA	f1 f2 f3 f4 f5 f6
RTP header	21 22 23 24 25 26 27 28 29 2a 2b 2c
Voice data	31 32 33 34 35 36 37 38 39 3a
CRC	93 86 b3 b9

PHS has removed the Type/Len field that would otherwise be included in the Ethernet/802.3 header. The User Data consists of the RTP header and the voice data.

Encryption is applied beginning with the first octet of the RTP header and ending with the last octet of the CRC, as follows:

Mode	CBC
Key	e6 60 0f d8 85 2e f5 ab
IV	81 0e 52 8e 1c 5f da 1a
Plaintext	21 22 23 24 25 26 27 28 29 2a 2b 2c 31 32 33 34 35 36 37 38 39 3a 93 86
Ciphertext	b4 55 da c8 39 1e 0c ed 15 cf b5 79 0a c3 24 5e cf 0f 52 c0 69 f5 f6 6e

Mode	CFB64
Key	e6 60 0f d8 85 2e f5 ab
IV	cf 0f 52 c0 69 f5 f6 6e
Plaintext	b3 b9 00 00 00 00 00 00
Ciphertext	3e 31 de ea 96 6a 88 6b

The PDU, after encryption, looks like this:

DA	01 02 03 04 05 06
SA	f1 f2 f3 f4 f5 f6
RTP header	b4 55 da c8 39 1e 0c ed 15 cf b5 79
Voice data	0a c3 24 5e cf 0f 52 c0 69 f5
CRC	f6 6e 3e 31

I.8.2 Upstream

Suppose that the PDU, after PHS and prior to encryption, is as follows:

RTP header	21 22 23 24 25 26 27 28 29 2a 2b 2c
Voice data	31 32 33 34 35 36 37 38 39 3a
CRC	65 cf fe 89

PHS has removed the DA, SA, and Type/Len fields that would otherwise be included in the Ethernet/802.3 header. The User Data consists of the RTP header and the voice data. The first 12 octets of the User Data are not encrypted.

Encryption is applied beginning with the first octet of the voice data and ending with the last octet of the CRC, as follows:

Mode	CBC
Key	e6 60 0f d8 85 2e f5 ab
IV	81 0e 52 8e 1c 5f da 1a
Plaintext	31 32 33 34 35 36 37 38
Ciphertext	d6 88 87 66 1f 66 04 79

Mode	CFB64
Key	e6 60 0f d8 85 2e f5 ab
IV	d6 88 87 66 1f 66 04 79
Plaintext	39 3a 65 cf fe 89 00 00
Ciphertext	c0 07 20 8e 3b 0b b1 b9

The PDU, after encryption, looks like this:

RTP header	21 22 23 24 25 26 27 28 29 2a 2b 2c
Voice data	d6 88 87 66 1f 66 04 79 c0 07
CRC	20 8e 3b 0b

I.9 Fragmented Packet Encryption (DES)

When a packet is fragmented, each fragment is independently encrypted using CBC mode with residual block processing. The TEK and IV for each fragment are the same TEK and IV that are used for encrypting an unfragmented PDU. All octets of a fragment are encrypted, including the 12 octets carrying the Ethernet/802.3 destination and source addresses (DA/SA) of the Packet PDU.

In the example here, no effort is made to use meaningful values for the fields of the packet. As a result, the example here is not a valid packet suitable for transmission. The intent of the example is to illustrate encryption details only.

In this example, the TEK and IV are taken from the example Key Reply packet described above.

Suppose that packet is divided into two fragments, as follows:

Fragment 1 payload	01 02 03 04 05 06 f1 f2 f3 f4 f5 f6 00 01 02 03 04 05
Fragment 1 CRC	b4 2b 6d d4

Fragment 2 payload	06 07 08 09 0a 0b 0c 0d
Fragment 2 CRC	48 34 45 36

The first fragment is encrypted using DES-CBC and DES-CFB64, as follows:

Mode	CBC
Key	e6 60 0f d8 85 2e f5 ab
IV	81 0e 52 8e 1c 5f da 1a
Plaintext	01 02 03 04 05 06 f1 f2 f3 f4 f5 f6 00 01 02 03
Ciphertext	47 41 0f 4f fd 78 47 6e c8 1a 67 4e 26 0c 20 c5

Mode	CFB64
Key	e6 60 0f d8 85 2e f5 ab
IV	c8 1a 67 4e 26 0c 20 c5
Plaintext	04 05 b4 2b 6d d4 00 00
Ciphertext	56 6d 5c 58 2f 56 dc 39

The first fragment, after encryption, looks like this:

Fragment 1 payload	47 41 0f 4f fd 78 47 6e c8 1a 67 4e 26 0c 20 c5 56 6d
Fragment 1 CRC	5c 58 2f 56

The second fragment is encrypted using DES-CBC and DES-CFB64, as follows:

Mode	CBC
Key	e6 60 0f d8 85 2e f5 ab
IV	81 0e 52 8e 1c 5f da 1a
Plaintext	06 07 08 09 0a 0b 0c 0d
Ciphertext	d8 55 0f 59 9d 19 d9 c6

Mode	CFB64
Key	e6 60 0f d8 85 2e f5 ab
IV	d8 55 0f 59 9d 19 d9 c6
Plaintext	48 34 45 36 00 00 00 00
Ciphertext	b4 5f 3e 95 0e e4 d7 df

The second fragment, after encryption, looks like this:

Fragment 2 payload	d8 55 0f 59 9d 19 d9 c6
Fragment 2 CRC	b4 5f 3e 95

I.10 Packet PDU Encryption (AES)

The first 12 octets of the Packet PDU, containing the Ethernet/802.3 destination and source addresses (DA/SA), are not encrypted. The remaining octets of the Packet PDU are encrypted in this example using AES-CBC mode with special handling of residual termination blocks that are less than 128 bits. The combination of AES-CBC and residual block processing ensures that the encryption does not change the length of the packet. The encryption key is the TEK corresponding to the key sequence number of the packet's Privacy Extended Header.

This specification describes the residual block processing as follows:

> The next-to-last ciphertext block is encrypted a second time, using the ECB mode of the encryption algorithm, and the least significant *n* bits of the result are XORed with the final *n* bits of the payload to generate the short final cipher block. In order for the receiver to decrypt the short final cipher block, the receiver encrypts the next-to-last ciphertext block using the ECB mode of the encryption algorithm, and XORs the left-most *n* bits with the short final cipher block in order to recover the short final cleartext block.

An alternative description of this procedure, which is equivalent to the description above, is as follows:

> Given a final block having *n* bits, where *n* is less than the length of the defined block for the cipher, the *n* bits are padded up to a block of the correct length by appending bits of arbitrary value to the right of the *n* payload bits. The resulting block is encrypted using the CFB<n> mode (where <n> is the length, in bits, of the block for the cipher in question) with the next-to-last ciphertext block serving as initialization vector for the CFB<n> operation. The leftmost *n* bits of the resulting ciphertext are used as the short cipher block. In the special case where the PDU is less than the length of the block for the cipher, the procedure is the same as for a short final block, with the provided initialization vector serving as the initialization vector for the CFB<n> operation.

The alternative description produces the same ciphertext as does the description in the body of this specification. In the alternative description, however, no mention is made of combining ECB encryption with XORs. These operations are internal to CFB, just as they are internal to CBC. The alternative description is convenient here because it allows residual block processing to be illustrated using CFB examples in [FIPS 46-3].

The Packet PDU includes the DA, SA, and Type/Len fields. In the examples here, no effort is made to use correct values for these fields. As a result, the examples here are not valid packets suitable for transmission. The intent of the examples is to illustrate encryption details only.

In these examples, the TEK and IV are taken from the example Key Reply packet described above.

I.10.1 CBC Only

When the number of octets to be encrypted is a multiple of 16, the encryption mode is AES-CBC as defined in [FIPS 46-3]. The encryption key and IV are as conveyed in the Key Reply packet.

The following table represents an example of AES-CBC encryption:

Mode	CBC
Key	01 23 45 67 89 ab cd ef 01 23 45 67 89 ab cd ef
IV	12 34 56 78 90 ab cd ef 12 34 56 78 90 ab cd ef
Example Plaintext	4e 6f 77 20 69 73 20 74 68 65 20 74 69 6d 65 20
Equivalent Ciphertext	7d 51 ac d5 b3 79 ac 2f 8f 46 f3 1c af d7 91 b4

Suppose that a PDU, prior to encryption, is as follows:

DA	01 02 03 04 05 06
SA	f1 f2 f3 f4 f5 f6
Type/Len	00 01
User Data	02 03 04 05 06 07 08 09 0a 0b
CRC	88 41 65 06

The DA and SA fields are not included in the encryption process, so the AES-CBC encryption is performed as follows:

Mode	CBC
Key	e6 60 0f d8 85 2e f5 ab e6 60 0f d8 85 2e f5 ab
IV	81 0e 52 8e 1c 5f da 1a 81 0e 52 8e 1c 5f da 1a
Plaintext	00 01 02 03 04 05 06 07 08 09 0a 0b 88 41 65 06
Ciphertext	5a 79 ba ca 6ª 2d 38 99 11 76 e3 11 9f f1 19 c7

The PDU, after encryption, looks like this:

DA	01 02 03 04 05 06
SA	f1 f2 f3 f4 f5 f6
Type/Len	5a 79
User Data	ba ca 6a 2d 38 99 11 76 e3 11
CRC	9f f1 19 c7

I.10.2 CBC with Residual Block Processing

When the number of octets to be encrypted is greater than 16 and is not a multiple of 16, the encryption mode is a combination of AES-CBC and AES-CFB128.

Encryption begins in AES-CBC mode. AES-CBC is used to process as many complete AES blocks as are present. The encryption key and IV are as conveyed in the Key Reply packet.

After the AES-CBC encryption, there is some number of octets that have not been encrypted. These octets are encrypted using AES-CFB128 mode. The encryption key is as in the Key Reply packet. The IV is the last 16 octets of ciphertext produced by the AES-CBC processing.

The following table represents an example of AES-CFB128 encryption:

Mode	AES-CFB128
Key	01 23 45 67 89 ab cd ef 01 23 45 67 89 ab cd ef
IV	12 34 56 78 90 ab cd ef 12 34 56 78 90 ab cd ef
Plaintext	4e 6f 77 20 69 73 20 74 68 65 20 74 69 6d 65 20
Ciphertext	43 bc 0a d0 fc 8d 93 ff 80 e0 bf f1 41 fc 67 08

Suppose that the PDU, prior to encryption, is as follows:

DA	01 02 03 04 05 06
SA	f1 f2 f3 f4 f5 f6
Type/Len	00 01
User Data	02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e
CRC	91 d2 d1 9f

The total number of octets to be encrypted is 19. The first 16 octets are processed using AES-CBC encryption, and the last 3 octets using AES-CFB128 encryption.

The AES-CBC encryption is performed as follows:

Mode	AES-CBC
Key	e6 60 0f d8 85 2e f5 ab e6 60 0f d8 85 2e f5 ab
IV	81 0e 52 8e 1c 5f da 1a 81 0e 52 8e 1c 5f da 1a
Plaintext	00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 91
Ciphertext	9d d1 67 4b ba 61 10 1b 56 75 64 74 36 4f 10 1d

The AES-CFB128 encryption is performed as follows:

Mode	CFB128
Key	e6 60 0f d8 85 2e f5 ab e6 60 0f d8 85 2e f5 ab
IV	9d d1 67 4b ba 61 10 1b 56 75 64 74 36 4f 10 1d
Plaintext	d2 d1 9f 00 00 00 00 00 00 00 00 00 00 00 00 00
Ciphertext	44 d4 73 dd 83 9c ee 46 4c ff 83 b7 27 96 d6 55

The key is the same as used for the AES-CBC encryption operation. The IV is the last 16 octets of ciphertext generated by the AES-CBC operation.

Notice that 13 octets of value 0 have been appended to the 3 plaintext octets. The values of these appended plaintext octets have no effect on the values of the first 3 ciphertext octets, which are the only ciphertext octets of interest. Arbitrary values can be used for the appended plaintext octets.

The PDU, after encryption, looks like this:

DA	01 02 03 04 05 06
SA	f1 f2 f3 f4 f5 f6
Type/Len	9d d1
User Data	67 4b ba 61 10 1b 56 75 64 74 36 4f 10
CRC	1d 44 d4 73

I.10.3 Runt Frame

When the number of octets to be encrypted is less than 16, the encryption mode is AES-CFB128. The encryption key and IV are as conveyed in the Key Reply packet.

Suppose that the PDU, prior to encryption, is as follows:

DA	01 02 03 04 05 06
SA	f1 f2 f3 f4 f5 f6
Type/Len	00 01
User Data	02
CRC	88 ee 59 7e

The AES-CFB128 encryption is performed as follows:

Mode	CFB128
Key	e6 60 0f d8 85 2e f5 ab e6 60 0f d8 85 2e f5 ab
IV	81 0e 52 8e 1c 5f da 1a 81 0e 52 8e 1c 5f da 1a
Plaintext	00 01 02 88 ee 59 7e 00 00 00 00 00 00 00 00 00
Ciphertext	fc 68 a3 55 60 37 dc d7 4c 6e 5c 5e 50 d5 98 b2

An octet of value 0 has been appended to the 7 plaintext octets. The value of this appended plaintext octet has no effect on the values of the first 7 ciphertext octets, which are the only ciphertext octets of interest. An arbitrary value can be used for the appended plaintext octet.

The PDU, after encryption, looks like this:

DA	01 02 03 04 05 06
SA	f1 f2 f3 f4 f5 f6
Type/Len	fc 68
User Data	a3
CRC	55 60 37 dc

I.11 Encryption of PDU with Payload Header Suppression (AES)

These examples show how encryption is applied to a PDU when Payload Header Suppression (PHS) is applied. The examples use an RTP [RFC 3550] Voice over IP payload with the Ethernet Type/Length field and the IP and UDP headers suppressed. In the examples, no effort is made to use correct values for the fields of the PDU. As a result, the examples here are not valid packets suitable for transmission. The intent of the examples is to illustrate encryption details only.

I.11.1 Downstream

Suppose that the PDU, after PHS and prior to encryption, is as follows:

DA	01 02 03 04 05 06
SA	f1 f2 f3 f4 f5 f6
RTP header	21 22 23 24 25 26 27 28 29 2a 2b 2c
Voice data	31 32 33 34 35 36 37 38 39 3a
CRC	93 86 b3 b9

PHS has removed the Type/Len field that would otherwise be included in the Ethernet/802.3 header. The User Data consists of the RTP header and the voice data. Encryption is applied beginning with the first octet of the RTP header and ending with the last octet of the CRC, as follows:

Mode	AES-CBC
Key	e6 60 0f d8 85 2e f5 ab e6 60 0f d8 85 2e f5 ab
IV	81 0e 52 8e 1c 5f da 1a 81 0e 52 8e 1c 5f da 1a
Plaintext	21 22 23 24 25 26 27 28 29 2a 2b 2c 31 32 33 34
Ciphertext	b6 d6 90 a5 8e 75 1d 00 9e 70 4f 1f 76 b1 5d 88

Mode	AES-CFB128
Key	e6 60 0f d8 85 2e f5 ab e6 60 0f d8 85 2e f5 ab
IV	b6 d6 90 a5 8e 75 1d 00 9e 70 4f 1f 76 b1 5d 88
Plaintext	35 36 37 38 39 3a 93 86 b3 b9 00 00 00 00 00 00
Ciphertext	26 25 03 a4 e5 01 d7 ab d9 9e 9a 63 00 22 44 36

The PDU, after encryption, looks like this:

DA	01 02 03 04 05 06
SA	f1 f2 f3 f4 f5 f6
RTP header	b6 d6 90 a5 8e 75 1d 00 9e 70 4f 1f
Voice data	76 b1 5d 88 af 4f a8 b4 ba 8a
CRC	6f 17 6b 7a

I.11.2 Upstream

Suppose that the PDU, after PHS and prior to encryption, is as follows:

RTP header	21 22 23 24 25 26 27 28 29 2a 2b 2c
Voice data	31 32 33 34 35 36 37 38 39 3a 3b 3c 3d 3e
CRC	65 cf fe 89

PHS has removed the DA, SA, and Type/Len fields that would otherwise be included in the Ethernet/802.3 header as well as the IP and UDP headers. The User Data consists of the RTP header and the voice data. The first 12 octets of the User Data are not encrypted.

Encryption is applied beginning with the first octet of the voice data and ending with the last octet of the CRC, as follows:

Mode	AES-CBC
Key	e6 60 0f d8 85 2e f5 ab e6 60 0f d8 85 2e f5 ab
IV	81 0e 52 8e 1c 5f da 1a 81 0e 52 8e 1c 5f da 1a
Plaintext	31 32 33 34 35 36 37 38 39 3a 3b 3c 3d 3e 65 cf
Ciphertext	ce 8c e2 55 1c e3 d2 6c 3f 06 f6 e9 66 e7 f7 d3

Mode	AES-CFB128
Key	e6 60 0f d8 85 2e f5 ab e6 60 0f d8 85 2e f5 ab
IV	ce 8c e2 55 1c e3 d2 6c 3f 06 f6 e9 66 e7 f7 d3
Plaintext	fe 89 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Ciphertext	4e 4e 1f c5 36 37 3e d7 bc 82 ad 52 8e 6c 05 dd

The PDU, after encryption, looks like this:

RTP header	21 22 23 24 25 26 27 28 29 2a 2b 2c
Voice data	ce 8c e2 55 1c e3 d2 6c 3f 06 f6 e9 66 e7
CRC	f7 d3 4e 4e

I.12 Fragmented Packet Encryption (AES)

When a packet is fragmented, each fragment is independently encrypted using CBC mode with residual block processing. The TEK and IV for each fragment are the same TEK and IV that are used for encrypting an unfragmented PDU. All octets of a fragment are encrypted, including the 12 octets carrying the Ethernet/802.3 destination and source addresses (DA/SA) of the Packet PDU.

In the example here, no effort is made to use meaningful values for the fields of the packet. As a result, the example here is not a valid packet suitable for transmission. The intent of the example is to illustrate encryption details only.

In this example, the TEK and IV are taken from the example Key Reply packet described above.

Suppose that packet is divided into two fragments, as follows:

Fragment 1 payload	01 02 03 04 05 06 f1 f2 f3 f4 f5 f6 00 01 02 03 04 05
Fragment 1 CRC	b4 2b 6d d4

Fragment 2 payload	06 07 08 09 0a 0b 0c 0d 06 07 08 09 0a 0b 0c 0d
Fragment 2 CRC	48 34 45 36

The first fragment is encrypted using AES-CBC and AES-CFB128, as follows:

Mode	AES-CBC
Key	e6 60 0f d8 85 2e f5 ab e6 60 0f d8 85 2e f5 ab
IV	81 0e 52 8e 1c 5f da 1a 81 0e 52 8e 1c 5f da 1a
Plaintext	01 02 03 04 05 06 f1 f2 f3 f4 f5 f6 00 01 02 03
Ciphertext	63 48 92 62 01 a9 88 08 df a3 55 30 7b 99 65 1e

Mode	AES-CFB128
Key	e6 60 0f d8 85 2e f5 ab e6 60 0f d8 85 2e f5 ab
IV	63 48 92 62 01 a9 88 08 df a3 55 30 7b 99 65 1e
Plaintext	04 05 b4 2b 6d d4 00 00 00 00 00 00 00 00 00 00
Ciphertext	e9 ad be e7 be d5 88 9d 35 ff 76 ce 29 26 98 04

The first fragment, after encryption, looks like this:

Fragment 1 payload	63 48 92 62 01 a9 88 08 df a3 55 30 7b 99 65 1e e9 ad
Fragment 1 CRC	be e7 be d5

The second fragment is encrypted using AES-CBC and AES-CFB128, as follows:

Mode	AES-CBC
Key	e6 60 0f d8 85 2e f5 ab e6 60 0f d8 85 2e f5 ab
IV	81 0e 52 8e 1c 5f da 1a 81 0e 52 8e 1c 5f da 1a
Plaintext	06 07 08 09 0a 0b 0c 0d 06 07 08 09 0a 0b 0c 0d
Ciphertext	62 f0 15 15 b4 18 d5 55 f7 52 5e 42 51 0b 77 e8

Mode	CFB128
Key	e6 60 0f d8 85 2e f5 ab e6 60 0f d8 85 2e f5 ab
IV	62 f0 15 15 b4 18 d5 55 f7 52 5e 42 51 0b 77 e8
Plaintext	48 34 45 36 00 00 00 00 00 00 00 00 00 00 00 00
Ciphertext	36 3f 5e 89 9a f7 40 89 85 7a 95 cd 47 d7 2b 1a

The second fragment, after encryption, looks like this:

Fragment 2 payload	62 f0 15 15 b4 18 d5 55 f7 52 5e 42 51 0b 77 e8
Fragment 2 CRC	36 3f 5e 89

I.13 Secure Software Download CM Code File

The code file example in this section was created using example versions of the Manufacturer CVC and CVC CA certificate.

Table 59 - New PKI Code File Example

	PKCS #7 Digital Signature {
0000: 30 82 0B 8D	ContentInfo header, len 2957(0xb8d)
0004: 06 09 2A 86 48 86 F7 0D 01 07 02	ContentType=signedData (OID: 1 2 840 113549 1 7 2)
0015: A0 82 0B 7E	[0] EXPLICIT, len 2942 (0xb7e)
0019: 30 82 0B 7A	SignedData header, len 2938 (0xb7a)
0023: 02 01 01	Version=1

0026: 31 0F
0028: 30 0D
0030: 06 09 60 86 48 01 65 03
04 02 01
0041: 05 00
	SET OF DigestAlgorithmIdentifier
	DigestAlgorithmIdentifier header
	Algorithm=SHA-256 (OID: 2 16 840 1 101 3 4 2 1)
	Parameters=NULL

0043: 30 0B	ContentInfo header

0045: 06 09 2A 86 48 86 F7 0D 01 07 01	contentType=data (OID: 1 2 840 113549 1 7 1)
0056: A0 82 09 43	Certificates Length = 2371(0x0943) Note: This field length includes the size(s) of all enclosed CVCs. If both Manufacturer and Co-signer CVCs are present, then this size is the total size of the signed Manufacturer CVC plus the total size of the signed Co-signer CVC.
0060: 30 82 03 F9 30 82 02 61 A0 03 02 01 02 02 08 05 05 05 05 05 05 05 05 30 0D 06 09 2A 86 48 86 F7 0D 01 01 0B 05 00 30 6C 31 0B 30 09 06 03 55 04 06 13 02 55 53 31 12 30 10 06 03 55 04 0A 13 09 43 61 62 6C 65 4C 61 62 73 31 11 30 0F 06 03 55 04 0B 13 08 43 56 43 20 43 41 30 31 31 36 30 34 06 03 55 04 03 13 2D 45 78 61 6D 70 6C 65 20 43 61 62 6C 65 4C 61 62 73 20 43 56 43 20 43 65 72 74 69 66 69 63 61 74 69 6F 6E 20 41 75 74 68 6F 72 69 74 79 30 1E 17 0D 31 34 30 39 31 37 31 38 32 31 35 34 5A 17 0D 32 34 30 39 31 37 31 38 32 31 35 34 5A 30 50 31 0B 30 09 06 03 55 04 06 13 02 55 53 31 11 30 0F 06 03 55 04 0A 13 08 42 72 6F 61 64 63 6F 6D 31 2E	Manufacturer Code Verification Certificate (CVC) (Required for all code files) Binary (DER) format CVC Begins at offset 60 (0x3C) CVC Ends at offset 1080(0x438) CVC Length 1021(0x3FD) See table I.13.2 for details of CVC.
30 2C 06 03 55 04 03 13 25 45 78 61 6D 70 6C 65 20 43 6F 64 65 20 56 65 72 69 66 69 63 61 74 69 6F 6E 20 43 65 72 74 69 66 69 63 61 74 65 30 82 01 22 30 0D 06 09 2A 86 48 86 F7 0D 01 01 01 05 00 03 82 01 0F 00 30 82 01 0A 02 82 01 01 00 E1 E7 F2 0A B4 B1 B6 A7 DD 0E 8E A7 6E DF B2 56 63 F4 6B 9C E9 40 D9 85 2B 5A 7B 98 4B F6 89 FE A8 39 7A FD F8 B8 D8 40 F0 C8 7E 0B 6F 9B 43 CD 28 99 7D 53 32 CD 6F D5 15 DD 2B ED 2D 4B B9 8F 05 A7 B6 C5 53 63 79 9B 76 8B 8F 9E DF 42 59 28 16 1B A8 EF BF 0C 36 28 3D 56 C4 FF 73 4B 56 6A 3A CE 27 36 12 33 B0 79 E5 13 6E 36 68 11 07 8D E3 99 69 22 F4 54 7C A2 CE 5A E2 38 73 2E 79 0E 17 AE 86 57 01 27 2E 50 13	CVC (continued)

BD 54 E5 36 4E 71 40 82 40 CB 8D 3B 03 92 5E 06 3F 20 DB 73 58 A3 72 3F 8F B8 F5 A5 D9 0D 22 99 DA 89 DC 2B A3 F4 6C 53 DD B2 06 73 CA 22 14 08 59 19 05 D6 92 49 56 5C 60 1D 9F 35 D2 53 DB E2 81 0D 94 B2 16 3B F1 AF 04 48 9C 41 54 07 43 18 65 DA C3 A2 A8 6C 2A A1 8D 16 51 55 16 EA 14 3C 5C B0 2F 8F 0A F9 4E 3E 7E 12 42 4B DD 7D 25 06 B6 0E 99 1E 7F 16 33 02 03 01 00 01 A3 3B 30 39 30 16 06 03 55 1D 25 01 01 FF 04 0C 30 0A 06 08 2B 06 01 05 05 07 03 03 30 1F 06 03 55 1D 23 04 18 30 16 80 14 BA 30 B0 DF 3A 7F 34 62 7C B7 A6 95 A3 0F 81 00 32 01 CC	
B0 30 0D 06 09 2A 86 48 86 F7 0D 01 01 0B 05 00 03 82 01 81 00 3F D3 6A F3 4B EA 89 02 C8 27 9C 43 AE 55 AF E5 7A A2 23 03 4A 29 69 78 D5 4D 4B AA E8 B2 7E 42 99 66 BC C9 82 A3 ED D2 F0 7F 77 22 E8 D7 C5 AD A0 58 DB 43 DA 6D 4B D6 1D 6E 73 38 0D 60 30 C0 9A 5F 22 62 7C BA F1 CE 44 B4 71 E6 72 D0 19 B5 5E 39 46 A9 13 00 18 7F 43 7A ED 31 A3 E5 7C FD 8B EE 4D AC 0D 46 09 59 BB 8C 6B A9 DA 44 62 35 65 4D 34 3E 41 A5 52 FE D2 A9 A2 FC 19 DF F3 70 9E 92 CD C2 24 32 BB 8A 58 45 12 17 B6 E2 AF 19 2D F5 3C 60 31 2F 71 87 F4 CA EC 7F C9 4B F4 3C 18 30 C5 B0 CF CC 45 EB C2 82 D9 3A 6D DD A4 8D E4 CD 7D 50 AA 7B FA 54 6A 36 D1 48 39 D7 D5 B2 AE 75 7D 76 DC F7 88 A5 F4 D6 C0 2D AD D6 80 5B 32 EC EA 47 20 44 A9 60 87 C4 8F 16 3F FC 83 48 B2 39 E8 46 FD 0A 91 AC C0 DC C7 7A 92 00 E0 58 0C CA 96 EB E0 BC 0F FA 9F 6A 19 D3 9E 0F 23 76 52 65 EE 5E 2B AB D8 F7 BC 9F D1 21 64 50 2C 42 C4 A3 51 47 7A 08 1E 19 BE 41 66 B0 1B 15 F7 65 5C 1C DC 3B 2F BE 37 F2 77 C2 A9	CVC (continued)

CA F6 93 73 66 1A 2E 16 03 28 84 CB A3 FA E5 FE 71 4C 07 32 54 61 20 D5 03 26 EF B7 38 ED 8E 36 38 43 6A 9A 05 1A DE 68 3A FB E9 A3 16 24 17 27 8F 39 61 49 63 DF 34 74 AB DE 3B 6F 3F FA 4E EC 50 D2 16 D4 5E CC 0A C3 07 F9 31 5A D6 AF E3 9F BD FA 98 8D 7F	
1081: 30 82 05 42 30 82 03 2A A0 03 02 01 02 02 08 04 04 04 04 04 04 04 04 30 0D 06 09 2A 86 48 86 F7 0D 01 01 0B 05 00 30 6E 31 0B 30 09 06 03 55 04 06 13 02 55 53 31 12 30 10 06 03 55 04 0A 13 09 43 61 62 6C 65 4C 61 62 73 31 12 30 10 06 03 55 04 0B 13 09 52 6F 6F 74 20 43 41 30 31 31 37 30 35 06 03 55 04 03 13 2E 45 78 61 6D 70 6C 65 20 43 61 62 6C 65 4C 61 62 73 20 52 6F 6F 74 20 43 65 72 74 69 66 69 63 61 74 69 6F 6E 20 41 75 74 68 6F 72 69 74 79 30 1E 17 0D 31 34 30 39 31 37 31 38 31 38 34 37 5A 17 0D 34 39 30 39 31 37 31 38 31 38 34 37 5A 30 6C 31 0B 30 09 06 03 55 04 06 13 02 55 53 31 12 30 10 06 03 55 04 0A 13 09 43 61 62 6C 65 4C 61 62 73 31 11 30 0F 06 03 55 04 0B 13 08 43 56 43 20 43 41 30 31 31 36 30 34 06 03 55 04 03 13 2D 45 78 61 6D 70 6C 65 20 43 61 62 6C 65 4C 61 62 73 20 43 56 43 20 43 65 72 74 69 66 69 63 61 74 69 6F 6E 20 41 75 74 68 6F 72 69 74 79 30 82 01 A2 30 0D 06 09 2A 86 48 86 F7 0D 01 01 01 05 00 03 82 01 8F 00 30 82 01 8A 02 82 01 81 00 B8 16 AB 6B 57 91 22 86 79 DF D9 8E 90 EC 04 A2 4D 5C A4 0B 7E 87 61 17 D2 8B 1A CF B3 D9 5B 7B B1 2A C8 57 90 2E F6 50 91 68 F9 42 18 94 90 C3 56 55 BC 80 39 D5 02 7D 29 CC E3 70 C1 F0 7F CD A0 09 8E 5C A3 AE DC 92 E7 32 8C 80 17 0E 33 B4 55 40 1F	CableLabs DOCSIS CVC CA Certificate (Required for all code files) Binary (DER) format CVC CA Begins at offset 1330 (0x532) CVC CA Ends at offset 2769 (0xAD1) CVC CA Length 1440 (0x5A0) See Table 63 for details of CVC CA.

87 9A 43 F1 B0 2F 07 6B F1 A6 BB 8B 51 E2 7A 9C 10 54 AC DC DA CE DB CC 89 5F 91 C6 57 7E 66 52 AC 77 8F 0D E6 29 3D 0D E5 F2 02 4A 88 AB 38 EB A4 8E 12 60 FC AC F2 A2 E5 70 F3 F8 31 CB BF BF FB 28 9A 56 4D C2 AF 6D A5 26 CA 3A DC 33 78 A7 D2 49 94 B8 84 F8 A2 AB BD AD 0B 32 5A 4B 1A 6F 61 FA 60 50 26 06 5D 5F 47 53 3A 50 73 A9 73 EE D1 8E 17 47 3A 1A 9B 0D C9 55 07 86 28 4F 8C BE 3E E0 19 98 66 07 F7 CD 20 F6 F5 2E 65 59 23 A8 EB 66 51 CA 15 1B 47 E3 5C 9B CB 62 C0 44 36 11 EC CF 14 7F 49 70 1B B6 B7 45 4D C4 99 B5 8E 78 DC 2D 70 4C FA 96 00 06 D2 38 00 35 8F AB 03 89 3B C4 66 FD EE 9E AD 30 F6 D2 CB 91 50 42 AE A0 54 61 C5 B9 02 4E EB 0C 00 E3 20 97 E5 80 F3 48 92 BB 30 DC 38 B9 18 19 05 C0 54 FB 0A A0 9F 4F 55 EB B9 22 CD 50 FC 96 8E 96 4A 91 00 67 19 3B 72 C0 31 62 23 05 EE 90 44 A1 40 3A 58 1D 17 65 7E 7E F7 2D C9 C6 E3 51 C4 F0 59 5B B2 48 31 FB CA 76 BA 6F 81 CE 9F 77 8A EB B7 A8 AD 02 03 01 00 01 A3 66 30 64 30 0E 06 03 55 1D 0F 01 01 FF 04 04 03 02 01 06 30 12 06 03 55 1D 13 01 01 FF 04 08 30 06 01 01 FF 02 01 00 30 1D 06 03 55 1D 0E 04 16 04 14 BA 30 B0 DF 3A 7F 34 62 7C B7 A6 95 A3 0F 81 00 32 01 CC B0 30 1F 06 03 55 1D 23 04 18 30 16 80 14 1F 43 E6 00 FC 16 2E 5A EA A5 4E 78 6F 04 64 E1 AE E6	CVC CA (continued)
A4 A8 30 0D 06 09 2A 86 48 86 F7 0D 01 01 0B 05 00 03 82 02 01 00 AC B2 7F 24 E6 B4 07 88 3F 6B ED 70 DA 05 AA 04 9C 62 F1 39 09 2B 3A BD DF 23 63 27 59 BB 01 40 81 72 F5 84 C2 79 C1 26 0B 14 F5 73 AA 21 BA 08 D1 8D 2C 59 7F E0 DC 7E 90 86 D0 11 45 BF F5 AB 26 21 0C F0 89 37 BD 51 AD 94	CVC CA (continued)

0C 90 10 E8 4B 7C C3 6E C7 41 03 13 43 F2 56 3E 1B 27 ED 86 15 9D 3A 03 72 ED 4B F0 6E 61 BD AD D1 09 00 04 66 83 8F C5 A2 C6 B7 F3 69 BF CA 5E E6 CA 56 70 21 25 10 1D A4 D9 6B BA 5F DD A1 49 53 E1 E2 4D CF 9D EF 44 5A 19 D6 B7 AB 74 E7 88 D6 6F 37 EB E0 2E F3 0C C5 FA 44 D9 2E F0 65 23 ED 67 2D A5 E9 D0 14 22 97 F6 84 CE 88 8F A3 71 1A CD 64 88 D9 F7 4B 61 83 FA 8D 20 12 01 A6 2B 13 34 2C 30 BD DD 0A 28 79 F1 C9 6A 34 4B 9B 50 22 A4 6A 12 B6 E0 CC 87 8F 35 46 FA E5 29 C8 E7 1A 2F 38 65 48 8D 20 E3 B2 86 23 AA 40 2B 76 93 9D 24 22 3D 6A A6 81 CE 12 4C FC 1A 03 49 21 33 33 C7 21 04 A2 20 FD D1 D4 C8 80 C3 58 A4 06 86 43 7B 65 3A 73 E5 B2 00 76 A2 20 4A FD 2C F5 97 45 67 D6 B4 52 49 93 A5 73 99 BB 49 86 B2 1A A0 6C 15 98 49 35 03 4A 66 75 ED B0 95 30 9C 67 7E 50 81 9D 0E B1 3F F2 A1 1B 7F 84 1E 95 86 52 99 69 70 8E B0 88 79 17 83 33 14 AD 29 72 4E 6E B4 C0 B9 5F D8 CD ED E1 CE 41 4A 49 BE 52 6E F1 6C 3D 21 02 47 49 9E B4 D6	
48 8C 8D 2D 5B CA B8 FC 03 DC D2 04 7A 50 01 37 AC FA EF 4B BE 14 62 60 D4 B5 1D 52 CB EB 64 BE 33 2A 5A 0E 04 D2 71 68 9D 6E 77 89 B6 62 A4 84 39 10 A6 8C F1 2D 9C 89 9E C1 4B 41 F6 D7 B9 28 61 AB 93 55 3B 4E 24 57 51 83 C5 FD BE 37 DD 4D D9 EA 7E 35 63 8B 3A 95 67 AA 2F C8 4E ED 81 22 B7 87 84 E6 80 0E 11 2D 61 4A F0 99 F6 A4 C9 5F F6 BA 1D 49 12 CF 5A 3A 77 4A 9A FA 1B 30	CVC CA (continued)
2431: 31 82 02 0E	SET OF SignerInfo header Note: This field length includes the size(s) of all enclosed SignerInfo structures. If both Manufacturer and Co-signer SignerInfo structures are present, then this size is the total size of the Manufacturer SignerInfo plus the total size of the Co-signer SignerInfo.

2435: 30 82 02 0A	Manufacturer SignerInfo header
2439 02 01 01	Version=1
2442: 30 78	IssuerAndSerialNumber header
2444: 30 6C	Issuer Name header

2446: 31 0B
2448: 30 09
2450: 06 03 55 04 06
2455: 13 02 55 53
	AttributeType=countryName
	AttributeValue="US"

2459: 31 12
2461: 30 10
2463: 06 03 55 04 0A
2468: 13 09 43 61 62 6C 65 4C
61 62 73
	AttributeType=organizationalName
	AttributeValue="CableLabs"

2479: 31 11
2481: 30 0F
2483: 06 03 55 04 0B
2488: 13 08 43 56 43 20 43 41
30 31
	AttributeType = organizationalUnitName
	AttributeValue = "CVC CA01"

2498: 31 36
2500: 30 34
2502: 06 03 55 04 03
2507: 13 2D 45 78 61 6D 70 6C
65 20 43 61 62 6C 65 4C
61 62 73 20 43 56 43 20
43 65 72 74 69 66 69 63
61 74 69 6F 6E 20 41 75
74 68 6F 72 69 74 79
	AttributeType=commonName
	AttributeValue= “Example CableLabs CVC Certification Authority”

2554: 02 08 05 05 05 05 05 05 05 05	CertificateSerialNumber = 05 05 05 05 05 05 05 05

2564: 30 0D
2566: 06 09 60 86 48 01 65 03
04 02 01
2577: 05 00
	DigestAlgorithmIdentifier header
	Algorithm=SHA-256 (OID: 2 16 840 1 101 3 4 2 1)
	Parameters=NULL

2579: A0 69	[0] AuthenticateAttributes

2581: 30 18
2583: 06 09 2A 86 48 86 F7 0D
01 09 03
2594: 31 0B
2596: 06 09 2A 86 48 86 F7 0D
01 07 01
	ContentType header
	AttributeType=contentType
	(OID: 1 2 840 113549 1 9 3)
	SET OF
	AttributeValue=Data
	(OID: 1 2 840 113549 1 7 1)

2607: 30 1C
2609: 06 09 2A 86 48 86 F7 0D
01 09 05
2620: 31 0F
2622: 17 0D 31 34 30 39 31 37
30 30 30 30 30 30 5A
	SigningTime header
	AttributeType= signingTime
	(pkcs-9 signingTime OID: 1 2 840 113549 1 9 5)
	SET OF
	attributeValue=2014/09/17/00:00:00Z GMT

2637: 30 2F
2639: 06 09 2A 86 48 86 F7 0D
01 09 04
	MessageDigest header
	AttributeType=messageDigest
	SET OF

2650: 31 22 2652: 04 20 C4 4E 77 9C B8 9C 12 AF 40 36 A8 CA E5 BF 27 C5 85 09 38 00 86 FB 07 A5 54 39 21 AA 2E 03 E4 B9	attributeValue=OCTET STRING Message Digest

2686: 30 0D
2688: 06 09 2A 86 48 86 F7 0D
01 01 01
2699: 05 00
	DigestEncryptionAlgorithm header
	Algorithm=RSA Encryption (OID: 1 2 840 113549 1 1 1)
	Parameters=NULL

2701: 04 82 01 00 A0 3F DC 85 B7 FD 38 D0 FF 82 32 B8 E5 1E 15 48 FB DE E0 F2 15 7C 8F B1 2F EB 18 71 EF 51 AD B7 58 08 EE 73 2D 2B F5 8D 08 44 F7 A0 9A 06 80 23 27 D7 5C C2 59 60 EC F0 D4 41 82 CA 83 9A F8 3A 61 6B 42 B6 79 51 CF 08 2B 13 AA DF 26 08 1F 43 17 18 71 7E 52 13 0C 6F 14 88 F0 75 89 2A AB B7 52 2B 04 70 AC D9 F1 6B 2E ED 7F 27 79 28 E4 7C 93 00 4D 45 90 88 1A 05 71 09 12 EF F0 82 AA C0 74 C1 7C 6F AE D0 98 50 75 FA 72 1E 0D 28 D5 5A 87 3E 49 95 57 91 07 01 07 0A B8 51 EE 5D 6C 54 39 C6 D6 12 A0 75 57 FC C0 50 E6 8D 91 BB F8 5E 35 C7 5C 59 0B DA 84 5F EA 7C F5 74 7A 9D BE 04 C4 2B CE 8A FD B3 86 2C 23 B0 1B D1 A8 0D 0E 8E A6 08 BA D9 A8 65 24 0D 14 69 C8 0D 04 4A 79 04 28 A5 23 77 F6 B1 1C 3B 17 8D 15 77 38 99 EC 7E 6C 03 A3 DD D8 4B FD CA 80 12 3C 1A 32 BD E8 B8	OCTET STRING EncryptedDigest
	End MfgSignerInfo
	End signer info
	End signed data
	End [PKCS#7] Digital Signature
	SignedContent{
2961: 1C 00 00	
	} End of Download Parameters

2964: 48 65 6C 6C 6F 20 57 6F
72 6C 64 0A
	CodeImage() {
	"Hello World\n"
	}
	Notes: Vendor-specific formatting

	} End of SignedContent

Table 60 - CVC Example

	X509 Certificate {
0000: 30 82 03 F9	CVC header

0004: 30 82 02 61	tbs CVC header
0008: A0 03 0010: 02 01 02	Version=v3
0013: 02 08 0015: 05 05 05 05 05 05 05 05	Serial Number = 05 05 05 05 05 05 05 05
0023: 30 0D 0025: 06 09 2A 86 48 86 F7 0D 01 01 0B 0036: 05 00	Signature Algorithm= SHA-256 with RSA Encryption (OID: 1 2 840 113549 1 1 11) Parameters = NULL
0038: 30 6C	Issuer SEQUENCE
0040: 31 0B 0042: 30 09 0044: 06 03 55 04 06 0049: 13 02 55 53	AttributeType= countryName (OID: 2 5 4 6) AttributeValue=”US”
0053: 31 12 0055: 30 10 0057: 06 03 55 04 0A 0062: 13 09 43 61 62 6C 65 4C 61 62 73	AttributeType= organizationalName (OID: 2 5 4 10) AttributeValue="CableLabs"
0073: 31 11 0075: 30 0F 0077: 06 03 55 04 0B 0082: 13 08 43 56 43 20 43 41 30 31	AttributeType= organizationalUnitName (OID: 2 5 4 11) AttributeValue="CVC CA01"
0092: 31 36 0094: 30 34 0096: 06 03 55 04 03 0101: 13 2D 45 78 61 6D 70 6C 65 20 43 61 62 6C 65 4C 61 62 73 20 43 56 43 20 43 65 72 74 69 66 69 63 61 74 69 6F 6E 20 41 75 74 68 6F 72 69 74 79	AttributeType= commonName (OID: 2 5 4 3) AttributeValue="Example CableLabs CVC Certification Authority"
0148: 30 1E 0150: 17 0D 0152: 31 34 30 39 31 37 31 38 32 31 35 34 5A 0165: 17 0D 32 34 30 39 31 37 31 38 32 31 35 34 5A	Validity SEQUENCE Not before=2014/09/17 18:21:54 GMT Not after=2024/09/17 18:21:54 GMT (10 Years)
0180: 30 50	Subject SEQUENCE

0182: 31 0B
0184: 30 09
0186: 06 03 55 04 06
0191: 13 02 55 53
	AttributeType=countryName
	(OID: 2 5 4 6)
	AttributeValue="US"

0195: 31 11 0197: 30 0F 0199: 06 03 55 04 0A 0204: 13 08 42 72 6F 61 64 63 6F 6D	AttributeType= organizationalName (OID: 2 5 4 10) AttributeValue="Broadcom"

0214: 31 2E
0216: 30 2C
0218: 06 03 55 04 03
0223: 13 25 45 78 61 6D 70 6C
65 20 43 6F 64 65 20 56
65 72 69 66 69 63 61 74
69 6F 6E 20 43 65 72 74
69 66 69 63 61 74 65
	AttributeType=commonName
	(OID: 2 5 4 3)
	AttributeValue="Code
	Verification Certificate"

0262: 30 82 01 22	SubjectPublicKeyInfo header

0266: 30 0D
0268: 06 09 2A 86 48 86 F7 0D 01 01 01
0279: 05 00
	Public Key SEQUENCE
	Algorithm=RSA encryption
	(OID: 1 2 840 113549 1 1 1)
	Parameters=NULL

0281: 03 82 01 0F 00	Public Key header

0286: 30 82 01 0A	Public Key modulus header
0290: 02 82 01 01 00 0295: E1 E7 F2 0A B4 B1 B6 A7 DD 0E 8E A7 6E DF B2 56 63 F4 6B 9C E9 40 D9 85 2B 5A 7B 98 4B F6 89 FE A8 39 7A FD F8 B8 D8 40 F0 C8 7E 0B 6F 9B 43 CD 28 99 7D 53 32 CD 6F D5 15 DD 2B ED 2D 4B B9 8F 05 A7 B6 C5 53 63 79 9B 76 8B 8F 9E DF 42 59 28 16 1B A8 EF BF 0C 36 28 3D 56 C4 FF 73 4B 56 6A 3A CE 27 36 12 33 B0 79 E5 13 6E 36 68 11 07 8D E3 99 69 22 F4 54 7C A2 CE 5A E2 38 73 2E 79 0E 17 AE 86 57 01 27 2E 50 13 BD 54 E5 36 4E 71 40 82 40 CB 8D 3B 03 92 5E 06 3F 20 DB 73 58 A3 72 3F 8F B8 F5 A5 D9 0D 22 99 DA 89 DC 2B A3 F4 6C 53 DD B2 06 73 CA 22 14 08 59 19 05 D6 92 49 56 5C 60 1D 9F 35 D2 53 DB E2 81 0D 94 B2 16 3B F1 AF 04 48 9C 41 54 07 43 18 65 DA C3 A2 A8 6C 2A A1 8D 16 51 55 16 EA 14 3C 5C B0 2F 8F 0A F9 4E 3E 7E 12 42 4B DD 7D 25 06 B6 0E 99 1E 7F 16 33	INTEGER Public Key modulus (Keysize=2048-bits)
0551: 02 03 01 00 01	INTEGER Public Key exponent=65537
0056: A3 3B 0058: 30 39 0560: 30 16 0562: 06 03 55 1D 25 0567: 01 01 FF 0570: 04 0C 0572: 30 0A 0574: 06 08 2B 06 01 05 05 07 03 03	[3] Extensions Extensions header SEQUENCE Extension header SEQUENCE Extended Key Usage (OID: 2 5 29 37) Critical=TRUE keyPurposeID=codeSigning (OID: 1 3 6 1 5 5 7 3 3)
0584: 30 1F 0586: 06 03 55 1D 23 0591: 04 18 0593: 30 16 0595: 80 14 BA 30 B0 DF 3A 7F 34 62 7C B7 A6 95 A3 0F 81 00 32 01 CC B0	Signature SEQUENCE Algorithm=SHA-1 with RSA (OID: 2 5 29 35) Parameters=NULL keyid: BA:30:B0:DF:3A:7F:34:62: 7C:B7:A6:95:A3:0F:81:00: 32:01:CC:B0
0632: 03 82 01 81 00	Signature value header
0637: 3F D3 6A F3 4B EA 89 02 C8 27 9C 43 AE 55 AF E5 7A A2 23 03 4A 29 69 78 D5 4D 4B AA E8 B2 7E 42 99 66 BC C9 82 A3 ED D2 F0 7F 77 22 E8 D7 C5 AD A0 58 DB 43 DA 6D 4B D6 1D 6E 73 38 0D 60 30 C0 9A 5F 22 62 7C BA F1 CE 44 B4 71 E6 72 D0 19 B5 5E 39 46 A9 13 00 18 7F 43 7A ED 31 A3 E5 7C FD 8B EE 4D AC 0D 46 09 59 BB 8C 6B A9 DA 44 62 35 65 4D 34 3E 41 A5 52 FE D2 A9 A2 FC 19 DF F3 70 9E 92 CD C2 24 32 BB 8A 58 45 12 17 B6 E2 AF 19 2D F5 3C 60 31 2F 71 87 F4 CA EC 7F C9 4B F4 3C 18 30 C5 B0 CF CC 45 EB C2 82 D9 3A 6D DD A4 8D E4 CD 7D 50 AA 7B FA 54 6A 36 D1 48 39 D7 D5 B2 AE 75 7D 76 DC F7 88 A5 F4 D6 C0 2D AD D6 80 5B 32 EC EA 47 20 44 A9 60 87 C4 8F 16 3F FC 83 48 B2 39 E8 46 FD 0A 91 AC C0 DC C7 7A 92 00 E0 58 0C CA 96 EB E0 BC 0F FA 9F 6A 19 D3 9E 0F 23 76 52 65 EE 5E 2B AB D8 F7 BC 9F D1 21 64 50 2C 42 C4 A3 51 47 7A 08 1E 19 BE 41 66 B0 1B 15 F7 65 5C 1C DC 3B 2F BE 37 F2 77 C2 A9 CA F6 93 73 66 1A 2E 16 03 28 84 CB A3 FA E5 FE 71 4C 07 32 54 61 20 D5 03 26 EF B7 38 ED 8E 36 38 43 6A 9A 05 1A DE 68 3A FB E9 A3 16 24 17 27 8F 39 61 49 63 DF 34 74 AB DE 3B 6F 3F FA 4E EC 50 D2 16 D4 5E CC 0A C3 07 F9 31 5A D6 AF E3 9F BD FA 98 8D 7F	Signature value

Table 61 - CVC-CA Example

	X509 Certificate {
0000: 30 82 05 42	CVC header
0004: 30 82 03 2A	tbs CVC header
0008: A0 03 0010: 02 01 02	Version=v3
0013: 02 08 04 04 04 04 04 04 04 04	Serial Number = 05 05 05 05 05 05 05 05
0023: 30 0D 0025: 06 09 2A 86 48 86 F7 0D 01 01 0B 0036: 05 00	Signature Algorithm=SHA-256 with RSA Encryption (OID: 1 2 840 113549 1 1 11) Parameters = NULL
0038: 30 6E	Issuer SEQUENCE

0040: 31 0B
0042: 30 09
0044: 06 03 55 04 06
0049: 13 02 55 53
	AttributeType=countryName
	(OID: 2 5 4 6)
	AttributeValue=”US”

0053: 31 12
0055: 30 10
0057: 06 03 55 04 0A
0062: 13 09 43 61 62 6C 65 4C 61 62 73
	AttributeType= organizationalName
	(OID: 2 5 4 10)
	AttributeValue="CableLabs"

0073: 31 12
0075: 30 10
0077: 06 03 55 04 0B
0082: 13 09 52 6F 6F 74 20 43 41 30 31
	AttributeType= organizationalUnitName
	(OID: 2 5 4 11)
	AttributeValue="Root CA01"

0093: 31 37
0095: 30 35
0097: 06 03 55 04 03
0102: 13 2E 45 78 61 6D 70 6C 65 20 43 61 62 6C 65 4C
61 62 73 20 52 6F 6F 74 20 43 65 72 74 69 66 69
63 61 74 69 6F 6E 20 41 75 74 68 6F 72 69 74 79
	AttributeType=commonName
	(OID: 2 5 4 3)
	AttributeValue="Example CableLabs Root Certification Authority"

0150: 30 1E 0152: 17 0D 31 34 30 39 31 37 31 38 31 38 34 37 5A 0167: 17 0D 34 39 30 39 31 37 31 38 31 38 34 37 5A	Validity SEQUENCE Not before= 2014/09/17 18:18:47 GMT Not after= 2049/09/17 18:18:47 GMT (35 Years)
0182: 30 6C	Subject SEQUENCE

0184: 31 0B
0186: 30 09
0188: 06 03 55 04 06
0193: 13 02 55 53
	AttributeType=countryName
	(OID: 2 5 4 6)
	AttributeValue="US"

0197: 31 12
0199: 30 10
0201: 06 03 55 04 0A
0206: 13 09 43 61 62 6C 65 4C 61 62 73
	AttributeType= organizationalName
	(OID: 2 5 4 10)
	AttributeValue="CableLabs"

0217: 31 11
0219: 30 0F
0221: 06 03 55 04 0B
0226: 13 08 43 56 43 20 43 41 30 31
	AttributeType= organizationalName
	(OID: 2 5 4 11)
	AttributeValue="CVC CA01"

0236: 31 36 0238: 30 34 0240: 06 03 55 04 03 0245: 13 2D 45 78 61 6D 70 6C 65 20 43 61 62 6C 65 4C 61 62 73 20 43 56 43 20 43 65 72 74 69 66 69 63 61 74 69 6F 6E 20 41 75 74 68 6F 72 69 74 79	AttributeType=commonName (OID: 2 5 4 3) AttributeValue= "CableLabs CVC Certification Authority"
0292: 30 82 01 A2	SubjectPublicKeyInfo header

0296: 30 0D
0298: 06 09 2A 86 48 86 F7 0D 01 01 01
0309: 05 00
	Public Key SEQUENCE
	Algorithm= RSA encryption
	(OID: 1 2 840 113549 1 1 1)
	Parameters=NULL

0311: 03 82 01 8F 00	Public Key header
0316: 30 82 01 8A	Public Key modulus header
0320: 02 82 01 81 00 0325: B8 16 AB 6B 57 91 22 86 79 DF D9 8E 90 EC 04 A2 4D 5C A4 0B 7E 87 61 17 D2 8B 1A CF B3 D9 5B 7B B1 2A C8 57 90 2E F6 50 91 68 F9 42 18 94 90 C3 56 55 BC 80 39 D5 02 7D 29 CC E3 70 C1 F0 7F CD A0 09 8E 5C A3 AE DC 92 E7 32 8C 80 17 0E 33 B4 55 40 1F 87 9A 43 F1 B0 2F 07 6B F1 A6 BB 8B 51 E2 7A 9C 10 54 AC DC DA CE DB CC 89 5F 91 C6 57 7E 66 52 AC 77 8F 0D E6 29 3D 0D E5 F2 02 4A 88 AB 38 EB A4 8E 12 60 FC AC F2 A2 E5 70 F3 F8 31 CB BF BF FB 28 9A 56 4D C2 AF 6D A5 26 CA 3A DC 33 78 A7 D2 49 94 B8 84 F8 A2 AB BD AD 0B 32 5A 4B 1A 6F 61 FA 60 50 26 06 5D 5F 47 53 3A 50 73 A9 73 EE D1 8E 17 47 3A 1A 9B 0D C9 55 07 86 28 4F 8C BE 3E E0 19 98 66 07 F7 CD 20 F6 F5 2E 65 59 23 A8 EB 66 51 CA 15 1B 47 E3 5C 9B CB 62 C0 44 36 11 EC CF 14 7F 49 70 1B B6 B7 45 4D C4 99 B5 8E 78 DC 2D 70 4C FA 96 00 06 D2 38 00 35 8F AB 03 89 3B C4 66 FD EE 9E AD 30 F6 D2 CB 91 50 42 AE A0 54 61 C5 B9 02 4E EB 0C 00 E3 20 97 E5 80 F3 48 92 BB 30 DC 38 B9 18 19 05 C0 54 FB 0A A0 9F 4F 55 EB B9 22 CD 50 FC 96 8E 96 4A 91 00 67 19 3B 72 C0 31 62 23 05 EE 90 44 A1 40 3A 58 1D 17 65 7E 7E F7 2D C9 C6 E3 51 C4 F0 59 5B B2 48 31 FB CA 76 BA 6F 81 CE 9F 77 8A EB B7 A8 AD	INTEGER Public Key modulus (Keysize=3072-bits)
0551: 02 03 0553: 01 00 01	INTEGER Public Key exponent=65537
0714: A3 66 0716: 30 64 0718: 30 0E 0720: 06 03 55 1D 0F 0725: 01 01 FF 0728: 04 04 0730: 03 02 01 06	[3] X509v3 Extensions Extensions header SEQUENCE Extension header SEQUENCE Key Usage (OID: 2 5 29 15) Critical=TRUE Certificate Sign CRL Sign
0734: 30 12 0736: 06 03 55 1D 13 0741: 01 01 FF 0744: 04 08 0746: 30 06 0748: 01 01 FF 0751: 02 01 00	Signature SEQUENCE basicConstraints (OID: 2 5 29 19) CA:TRUE pathlen:0
0754: 30 1D 0756: 06 03 55 1D 0E 0761: 04 16 0763: 04 14 BA 30 B0 DF 3A 7F 34 62 7C B7 A6 95 A3 0F 81 00 32 01 CC B0	subjectKeyIdentifier (OID: 2 5 29 14) BA:30:B0:DF:3A:7F:34:62: 7C:B7:A6:95:A3:0F:81:00: 32:01:CC:B0
0785: 30 1F 0787: 06 03 55 1D 23 0792: 04 18 0794: 30 16 0796: 80 14 1F 43 E6 00 FC 16 2E 5A EA A5 4E 78 6F 04 64 E1 AE E6 A4 A8	authorityKeyIdentifier (OID: 2 5 29 35) keyid: 1F:43:E6:00:FC:16:2E:5A: EA:A5:4E:78:6F:04:64:E1: AE:E6:A4:A8
0818: 30 0D 0820: 06 09 2A 86 48 86 F7 0D 01 01 0B 0831: 05 00	SHA-256 with RSA Encryption (OID: 1 2 840 113549 1 1 11)

0833: 03 82 02 01 00	Signature value header
0838: AC B2 7F 24 E6 B4 07 88 3F 6B ED 70 DA 05 AA 04 9C 62 F1 39 09 2B 3A BD DF 23 63 27 59 BB 01 40 81 72 F5 84 C2 79 C1 26 0B 14 F5 73 AA 21 BA 08 D1 8D 2C 59 7F E0 DC 7E 90 86 D0 11 45 BF F5 AB 26 21 0C F0 89 37 BD 51 AD 94 0C 90 10 E8 4B 7C C3 6E C7 41 03 13 43 F2 56 3E 1B 27 ED 86 15 9D 3A 03 72 ED 4B F0 6E 61 BD AD D1 09 00 04 66 83 8F C5 A2 C6 B7 F3 69 BF CA 5E E6 CA 56 70 21 25 10 1D A4 D9 6B BA 5F DD A1 49 53 E1 E2 4D CF 9D EF 44 5A 19 D6 B7 AB 74 E7 88 D6 6F 37 EB E0 2E F3 0C C5 FA 44 D9 2E F0 65 23 ED 67 2D A5 E9 D0 14 22 97 F6 84 CE 88 8F A3 71 1A CD 64 88 D9 F7 4B 61 83 FA 8D 20 12 01 A6 2B 13 34 2C 30 BD DD 0A 28 79 F1 C9 6A 34 4B 9B 50 22 A4 6A 12 B6 E0 CC 87 8F 35 46 FA E5 29 C8 E7 1A 2F 38 65 48 8D 20 E3 B2 86 23 AA 40 2B 76 93 9D 24 22 3D 6A A6 81 CE 12 4C FC 1A 03 49 21 33 33 C7 21 04 A2 20 FD D1 D4 C8 80 C3 58 A4 06 86 43 7B 65 3A 73 E5 B2 00 76 A2 20 4A FD 2C F5 97 45 67 D6 B4 52 49 93 A5 73 99 BB 49 86 B2 1A A0 6C 15 98 49 35 03 4A 66 75 ED B0 95 30 9C 67 7E 50 81 9D 0E B1 3F F2 A1 1B 7F 84 1E 95 86 52 99 69 70 8E B0 88 79 17 83 33 14 AD 29 72 4E 6E B4 C0 B9 5F D8 CD ED E1 CE 41 4A 49 BE 52 6E F1 6C 3D 21 02 47 49 9E B4 D6 48 8C 8D 2D 5B CA B8 FC 03 DC D2 04 7A 50 01 37 AC FA EF 4B BE 14 62 60 D4 B5 1D 52 CB EB 64 BE 33 2A 5A 0E 04 D2 71 68 9D 6E 77 89 B6 62 A4 84 39 10 A6 8C F1 2D 9C 89 9E C1 4B 41 F6 D7 B9 28 61 AB 93 55 3B 4E 24 57 51 83 C5 FD BE 37 DD 4D D9 EA 7E 35 63 8B 3A 95 67 AA 2F C8 4E ED 81 22 B7 87 84 E6 80 0E 11 2D 61 4A F0 99 F6 A4 C9 5F F6 BA 1D 49 12 CF 5A 3A 77 4A 9A FA 1B 30	Signature value

Appendix II Example of Multilinear Modular Hash (MMH) Algorithm Implementation (Informative)

This appendix gives an example implementation of the MMH MAC algorithm defined in Section 11.7.2. There may be other implementations that have advantages over this example in particular operating environments. This example is for informational purposes only and does not completely follow the DOCSIS MMH MAC algorithm defined in Section 11.7.3.

A main program is included for exercising the example implementation. The output produced by the program is included.

```
// Example code for MMH algorithm as defined in the DOCSIS 3.0 security specification

// Based on code in PacketCable security specification

// Original code by Mike Sabin
// DOCSIS modifications by Doc Evans, N7DR@arrisi.com
// This code is far from optimal; it is intended to be easy to follow, not fast to
execute

// Written for gcc 3.3.6

#include <iomanip>
#include <iostream>
#include <string>

#include <inttypes.h>
#include <stdio.h>

// Define this symbol to see intermediate values.
#define VERBOSE

// Define this symbol if you want to duplicate the PacketCable test vectors
#undef PACKETCABLE

// save ourselves some typing
using namespace std;

// Routine to reduce an int32_t value modulo F4, where F4 = 0x10001.
// Result is therefore in range [0, 0x10000].
const int32_t reduceModF4(const int32_t x)
{ int32_t rv = x;                            // holder for return value

// If x is negative, add a multiple of F4 to make it non-negative.
// This loop executes no more than twice.
  while (rv < 0)
    rv += 0x7fff7fff;

// Subtract high 16 bits of rv from low 16 bits.
  const int32_t xHi = rv >> 16;
  const int32_t xLo = rv & 0xffff;
  rv = xLo - xHi;

// If x is negative, add F4.
  if (rv < 0)
    rv += 0x10001;

// we are done
  return rv;
}
```

```
/*
Compute and return the MMH16 MAC of the message using the
indicated key and one-time pad.
The length of the key is to be at least msgLen bytes.
The length of the pad is at least two bytes.
*/
const uint16_t mmh16(const string& message,
                     const string& key,
                     const string& pad)
{
// check lengths
  if (key.length() < message.length())
    throw exception();

  if (pad.length() < 2)
    throw exception();

// ok to proceed
  int32_t sum = 0;        // 32-bit accumulator

  for (unsigned int i = 0; i < message.length(); i += 2)
  {
// Build a 16-bit factor from the next two message octets
    const int16_t x = (static_cast<uint16_t>(static_cast<unsigned char>(message[i]))
<< 8)
                     |
                       static_cast<uint16_t>(static_cast<unsigned char>(message[i +
1]));

// Build a 16-bit factor from the next two key octets
    const int16_t y = (static_cast<uint16_t>(static_cast<unsigned char>(key[i])) << 8)
|
                       static_cast<uint16_t>(static_cast<unsigned char>(key[i + 1]));

// Accumulate product of the factors into 32-bit sum
    sum += (static_cast<int32_t>(x) * static_cast<int32_t>(y));

#if defined(VERBOSE)
    cout << hex << " x 0x" << setw(2) << x
            << " y 0x" << setw(2) << y
            << " sum 0x" << setw(2) << sum << dec << endl;
#endif    // VERBOSE
  }

// Reduce sum modulo F4 and truncate to 16 bits
  uint16_t u = static_cast<uint16_t>(reduceModF4(sum));

#if defined(VERBOSE)
  cout << hex << " sum mod F4, truncated to 16 bits: 0x" << setw(2) << u << dec <<
endl;
#endif    // VERBOSE

// Build the pad variable from the two pad bytes
  const uint16_t v = (static_cast<uint16_t>(static_cast<unsigned char>(pad[0])) << 8)
|
                      static_cast<uint16_t>(static_cast<unsigned char>(pad[1]));

#if defined(VERBOSE)
  cout << hex << " pad variable: 0x" << setw(2) << v << dec << endl;
#endif    // VERBOSE

/* Accumulate pad variable, truncate to 16 bits */
  u = static_cast<uint16_t>(u + v);
#if defined(VERBOSE)
  cout << hex << " mmh16 value: 0x" << setw(2) << u << dec << endl;
#endif    // VERBOSE
  return u;
}

/*
Compute and return the MMH32 MAC of the message using the
indicated key and pad.
The length of the message is msgLen bytes; msgLen is to be even.
The length of the key is to be at least (msgLen + 2) bytes.
The length of the pad is four bytes. The pad is to be freshly
picked from a secure random source.
*/
const uint32_t mmh32(const string& message,
                     const string& key,
                     const string& pad)
{
// check lengths
  if (key.length() < (message.length() + 2))
    throw exception();

  if (pad.length() < 4)
    throw exception();

  const uint16_t x = mmh16(message, key, pad);
  const uint16_t y = mmh16(message, key.substr(2), pad.substr(2));
  const uint32_t sum = (static_cast<const uint32_t>(x) << 16) |
                        static_cast<const uint32_t>(y);

  return sum;
}

/*
Compute and return the MMH64 MAC of the message using the
indicated key and pad.
The length of the message is msgLen bytes; msgLen is to be even.
The length of the key is to be at least (msgLen + 6) bytes.
The length of the pad is eight bytes. The pad is to be freshly
picked from a secure random source.
*/
const uint64_t mmh64(const string& message,
                     const string& key,
                     const string& pad)
{
// check lengths
  if (key.length() < (message.length() + 6))
    throw exception();

  if (pad.length() < 8)
    throw exception();

  const uint16_t a = mmh16(message, key, pad);
  const uint16_t b = mmh16(message, key.substr(2), pad.substr(2));
  const uint16_t c = mmh16(message, key.substr(4), pad.substr(4));
  const uint16_t d = mmh16(message, key.substr(6), pad.substr(6));

  const uint64_t sum = ((((static_cast<const uint64_t>(a) << 16) |
                            static_cast<const uint64_t>(b)) << 16) |
                        static_cast<const uint64_t>(c)) << 16) |
                        static_cast<const uint64_t>(d);
  return sum;
}

/*
Routine to display a byte array
*/
template<class T>
void show(const string& rubric, const T& src, const unsigned int len)
{ const unsigned int BYTES_PER_LINE = 16;

  cout << rubric;

  for (unsigned int i = 0; i < len; i++)
  { if ((i % BYTES_PER_LINE) == 0)
      cout << endl;

    cout << setw(2) << hex << (unsigned int)(unsigned char)(src[i]) << " ";
  }

  cout << endl;
}

// helper function for displaying results
template<class T>
const string output_octets(const T& src)
{ string rv;
  const unsigned char* cp = (const unsigned char*)(&src);

  for (int n = sizeof(src) - 1; n >= 0; n--)
    rv += cp[n];

  return rv;
}

// example of use
int main(void)
{
// set formatting for cout
  cout.fill('0');

  string key_;
  string pad_;

// define some test vectors

#if !defined(PACKETCABLE)
// a trivial keystream
  for (unsigned int key_nr = 0; key_nr < 100; key_nr++)
    key_ += char(key_nr);

// a similarly trivial one-time pad (that is much longer than we need);
  for (unsigned int pad_nr = 0; pad_nr < 100; pad_nr++)
    pad_ += char(pad_nr + 1);

// an historically interesting message that is to be hashed
  const string message_("The Magic Words are Squeamish Ossifrage");
#endif     // !PACKETCABLE

#if defined(PACKETCABLE)
  const string message_("Now is the time.");

  unsigned char key[] = {
  0x35, 0x2c, 0xcf, 0x84, 0x95, 0xef, 0xd7, 0xdf, 0xb8,
  0xf5, 0x74, 0x05, 0x95, 0xeb, 0x98, 0xd6, 0xeb, 0x98,
  };

  unsigned char pad16[] = {
  0xae, 0x07,
  };

  unsigned char pad32[] = {
  0xbd, 0xe1, 0x89, 0x7b,
  };

  for (unsigned int key_nr = 0; key_nr < sizeof(key); key_nr++)
    key_ += key[key_nr];

  for (unsigned int pad_nr = 0; pad_nr < sizeof(pad16); pad_nr++)
    pad_ += pad16[pad_nr];
#endif    // PACKETCABLE

// MMH16
  cout << "Example of MMH16 computation" << endl;

  show("message", message_, message_.length());
  show("key", key_, message_.length());
  show("pad", pad_, 2);

  const uint16_t mac16 = mmh16(message_, key_, pad_);

  show("MMH16 MAC", output_octets(mac16), sizeof(mac16));
  cout << endl;

#if defined(PACKETCABLE)
  pad_.clear();
  for (unsigned int apad_nr = 0; apad_nr < sizeof(pad32); apad_nr++)
    pad_ += pad32[apad_nr];
#endif    // PACKETCABLE

// MMH32
  cout << "Example of MMH32 computation" << endl;

  show("message", message_, message_.length());
  show("key", key_, message_.length() + 2);
  show("pad", pad_, 4);

  const uint32_t mac32 = mmh32(message_, key_, pad_);

  show("MMH32 MAC", output_octets(mac32), sizeof(mac32));
  cout << endl;

#if !defined(PACKETCABLE)
// MMH64
  cout << "Example of MMH64 computation" << endl;

  show("message", message_, message_.length());
  show("key", key_, message_.length() + 6);
  show("pad", pad_, 8);

  const uint64_t mac64 = mmh64(message_, key_, pad_);

  show("MMH64 MAC", output_octets(mac64), sizeof(mac64));
  cout << endl;
#endif    // !PACKETCABLE

  return 0;
```

```
}


/*
Here is the VERBOSE output produced by the program if PACKETCABLE is defined:

Example of MMH16 computation
message
4e 6f 77 20 69 73 20 74 68 65 20 74 69 6d 65 2e
key
35 2c cf 84 95 ef d7 df b8 f5 74 05 95 eb 98 d6
pad
ae 07
 x 0x4e6f y 0x352c sum 0x104a7614
 x 0x7720 y 0xcf84 sum 0xf9bac294
 x 0x6973 y 0x95ef sum 0xce0a23f1
 x 0x2074 y 0xd7df sum 0xc8f3d4fd
 x 0x6865 y 0xb8f5 sum 0xabfb55a6
 x 0x2074 y 0x7405 sum 0xbab087ea
 x 0x696d y 0x95eb sum 0x8f00bff9
 x 0x652e y 0x98d6 sum 0x663aa46d
 sum mod F4, truncated to 16 bits: 0x3e33
 pad variable: 0xae07
 mmh16 value: 0xec3a
MMH16 MAC
ec 3a

Example of MMH32 computation
message
4e 6f 77 20 69 73 20 74 68 65 20 74 69 6d 65 2e
key
35 2c cf 84 95 ef d7 df b8 f5 74 05 95 eb 98 d6
eb 98
pad
bd e1 89 7b
 x 0x4e6f y 0x352c sum 0x104a7614
 x 0x7720 y 0xcf84 sum 0xf9bac294
 x 0x6973 y 0x95ef sum 0xce0a23f1
 x 0x2074 y 0xd7df sum 0xc8f3d4fd
 x 0x6865 y 0xb8f5 sum 0xabfb55a6
 x 0x2074 y 0x7405 sum 0xbab087ea
 x 0x696d y 0x95eb sum 0x8f00bff9
 x 0x652e y 0x98d6 sum 0x663aa46d
 sum mod F4, truncated to 16 bits: 0x3e33
 pad variable: 0xbde1
 mmh16 value: 0xfc14
 x 0x4e6f y 0xcf84 sum 0xf125323c
 x 0x7720 y 0x95ef sum 0xbfca091c
 x 0x6973 y 0xd7df sum 0xaf427949
 x 0x2074 y 0xb8f5 sum 0xa640e84d
 x 0x6865 y 0x7405 sum 0xd590b646
 x 0x2074 y 0x95eb sum 0xc81e04c2
 x 0x696d y 0x98d6 sum 0x9da1dde0
 x 0x652e y 0xeb98 sum 0x95912b30
 sum mod F4, truncated to 16 bits: 0x959f
 pad variable: 0x897b
 mmh16 value: 0x1f1a
MMH32 MAC
fc 14 1f 1a


Here is the VERBOSE output produced by the program if PACKETCABLE is not defined:
```

```
Example of MMH16 computation
message
54 68 65 20 4d 61 67 69 63 20 57 6f 72 64 73 20
61 72 65 20 53 71 75 65 61 6d 69 73 68 20 4f 73
73 69 66 72 61 67 65
key
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f
20 21 22 23 24 25 26
pad
01 02
 x 0x5468 y 0x01 sum 0x5468
 x 0x6520 y 0x203 sum 0xcbc3c8
 x 0x4d61 y 0x405 sum 0x202caad
 x 0x6769 y 0x607 sum 0x472148c
 x 0x6320 y 0x809 sum 0x78e90ac
 x 0x576f y 0xa0b sum 0xafca871
 x 0x7264 y 0xc0d sum 0x105f2785
 x 0x7320 y 0xe0f sum 0x16b1a665
 x 0x6172 y 0x1011 sum 0x1ccf3ef7
 x 0x6520 y 0x1213 sum 0x23f30057
 x 0x5371 y 0x1415 sum 0x2a7eac9c
 x 0x7565 y 0x1617 sum 0x349fe6af
 x 0x616d y 0x1819 sum 0x3dcba254
 x 0x6973 y 0x1a1b sum 0x488c6f75
 x 0x6820 y 0x1c1d sum 0x53fbbb15
 x 0x4f73 y 0x1e1f sum 0x5d54d402
 x 0x7369 y 0x2021 sum 0x6bd0d48b
 x 0x6672 y 0x2223 sum 0x7979fa21
 x 0x6167 y 0x2425 sum 0x873a8a04
 x 0x6500 y 0x2627 sum 0x9647ed04
 sum mod F4, truncated to 16 bits: 0x56bc
 pad variable: 0x102
 mmh16 value: 0x57be
MMH16 MAC
57 be

Example of MMH32 computation
message
54 68 65 20 4d 61 67 69 63 20 57 6f 72 64 73 20
61 72 65 20 53 71 75 65 61 6d 69 73 68 20 4f 73
73 69 66 72 61 67 65
key
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f
20 21 22 23 24 25 26 27 28
pad
01 02 03 04
 x 0x5468 y 0x01 sum 0x5468
 x 0x6520 y 0x203 sum 0xcbc3c8
 x 0x4d61 y 0x405 sum 0x202caad
 x 0x6769 y 0x607 sum 0x472148c
 x 0x6320 y 0x809 sum 0x78e90ac
 x 0x576f y 0xa0b sum 0xafca871
 x 0x7264 y 0xc0d sum 0x105f2785
 x 0x7320 y 0xe0f sum 0x16b1a665
 x 0x6172 y 0x1011 sum 0x1ccf3ef7
 x 0x6520 y 0x1213 sum 0x23f30057
 x 0x5371 y 0x1415 sum 0x2a7eac9c
 x 0x7565 y 0x1617 sum 0x349fe6af
 x 0x616d y 0x1819 sum 0x3dcba254
 x 0x6973 y 0x1a1b sum 0x488c6f75
 x 0x6820 y 0x1c1d sum 0x53fbbb15
 x 0x4f73 y 0x1e1f sum 0x5d54d402
 x 0x7369 y 0x2021 sum 0x6bd0d48b
 x 0x6672 y 0x2223 sum 0x7979fa21
 x 0x6167 y 0x2425 sum 0x873a8a04
 x 0x6500 y 0x2627 sum 0x9647ed04
 sum mod F4, truncated to 16 bits: 0x56bc
 pad variable: 0x102
 mmh16 value: 0x57be
 x 0x5468 y 0x203 sum 0xa9cd38
 x 0x6520 y 0x405 sum 0x24046d8
 x 0x4d61 y 0x607 sum 0x412aa7f
 x 0x6769 y 0x809 sum 0x7519530
 x 0x6320 y 0xa0b sum 0xb351790
 x 0x576f y 0xc0d sum 0xf52bc33
 x 0x7264 y 0xe0f sum 0x159ae80f
 x 0x7320 y 0x1011 sum 0x1cd48d2f
 x 0x6172 y 0x1213 sum 0x23b5cca5
 x 0x6520 y 0x1415 sum 0x2ba49845
 x 0x5371 y 0x1617 sum 0x32d7cd6c
 x 0x7565 y 0x1819 sum 0x3de4bc49
 x 0x616d y 0x1a1b sum 0x47d414c8
 x 0x6973 y 0x1c1d sum 0x53689acf
 x 0x6820 y 0x1e1f sum 0x5fa8f6af
 x 0x4f73 y 0x2021 sum 0x69a19482
 x 0x7369 y 0x2223 sum 0x79054ddd
 x 0x6672 y 0x2425 sum 0x877c2457
 x 0x6167 y 0x2627 sum 0x96004508
 x 0x6500 y 0x2829 sum 0xa5d87208
 sum mod F4, truncated to 16 bits: 0xcc30
 pad variable: 0x304
 mmh16 value: 0xcf34
MMH32 MAC
57 be cf 34

Example of MMH64 computation
message
54 68 65 20 4d 61 67 69 63 20 57 6f 72 64 73 20
61 72 65 20 53 71 75 65 61 6d 69 73 68 20 4f 73
73 69 66 72 61 67 65
key
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f
20 21 22 23 24 25 26 27 28 29 2a 2b 2c
pad
01 02 03 04 05 06 07 08
 x 0x5468 y 0x01 sum 0x5468
 x 0x6520 y 0x203 sum 0xcbc3c8
 x 0x4d61 y 0x405 sum 0x202caad
 x 0x6769 y 0x607 sum 0x472148c
 x 0x6320 y 0x809 sum 0x78e90ac
 x 0x576f y 0xa0b sum 0xafca871
 x 0x7264 y 0xc0d sum 0x105f2785
 x 0x7320 y 0xe0f sum 0x16b1a665
 x 0x6172 y 0x1011 sum 0x1ccf3ef7
 x 0x6520 y 0x1213 sum 0x23f30057
 x 0x5371 y 0x1415 sum 0x2a7eac9c
 x 0x7565 y 0x1617 sum 0x349fe6af
 x 0x616d y 0x1819 sum 0x3dcba254
 x 0x6973 y 0x1a1b sum 0x488c6f75
 x 0x6820 y 0x1c1d sum 0x53fbbb15
 x 0x4f73 y 0x1e1f sum 0x5d54d402
 x 0x7369 y 0x2021 sum 0x6bd0d48b
 x 0x6672 y 0x2223 sum 0x7979fa21
```

```
x 0x6167 y 0x2425 sum 0x873a8a04
x 0x6500 y 0x2627 sum 0x9647ed04
sum mod F4, truncated to 16 bits: 0x56bc
pad variable: 0x102
mmh16 value: 0x57be
x 0x5468 y 0x203 sum 0xa9cd38
x 0x6520 y 0x405 sum 0x24046d8
x 0x4d61 y 0x607 sum 0x412aa7f
x 0x6769 y 0x809 sum 0x7519530
x 0x6320 y 0xa0b sum 0xb351790
x 0x576f y 0xc0d sum 0xf52bc33
x 0x7264 y 0xe0f sum 0x159ae80f
x 0x7320 y 0x1011 sum 0x1cd48d2f
x 0x6172 y 0x1213 sum 0x23b5cca5
x 0x6520 y 0x1415 sum 0x2ba49845
x 0x5371 y 0x1617 sum 0x32d7cd6c
x 0x7565 y 0x1819 sum 0x3de4bc49
x 0x616d y 0x1a1b sum 0x47d414c8
x 0x6973 y 0x1c1d sum 0x53689acf
x 0x6820 y 0x1e1f sum 0x5fa8f6af
x 0x4f73 y 0x2021 sum 0x69a19482
x 0x7369 y 0x2223 sum 0x79054ddd
x 0x6672 y 0x2425 sum 0x877c2457
x 0x6167 y 0x2627 sum 0x96004508
x 0x6500 y 0x2829 sum 0xa5d87208
sum mod F4, truncated to 16 bits: 0xcc30
pad variable: 0x304
mmh16 value: 0xcf34
x 0x5468 y 0x405 sum 0x1534608
x 0x6520 y 0x607 sum 0x3b4c9e8
x 0x4d61 y 0x809 sum 0x6228a51
x 0x6769 y 0xa0b sum 0xa3115d4
x 0x6320 y 0xc0d sum 0xedb9e74
x 0x576f y 0xe0f sum 0x13a8cff5
x 0x7264 y 0x1011 sum 0x1ad6a899
x 0x7320 y 0x1213 sum 0x22f773f9
x 0x6172 y 0x1415 sum 0x2a9c5a53
x 0x6520 y 0x1617 sum 0x33563033
x 0x5371 y 0x1819 sum 0x3b30ee3c
x 0x7565 y 0x1a1b sum 0x472991e3
x 0x616d y 0x1c1d sum 0x51dc873c
x 0x6973 y 0x1e1f sum 0x5e44c629
x 0x6820 y 0x2021 sum 0x6b563249
x 0x4f73 y 0x2223 sum 0x75ee5502
x 0x7369 y 0x2425 sum 0x8639c72f
x 0x6672 y 0x2627 sum 0x957e4e8d
x 0x6167 y 0x2829 sum 0xa4c6000c
x 0x6500 y 0x2a2b sum 0xb568f70c
sum mod F4, truncated to 16 bits: 0x41a3
pad variable: 0x506
mmh16 value: 0x46a9
x 0x5468 y 0x607 sum 0x1fcbed8
x 0x6520 y 0x809 sum 0x5294cf8
x 0x4d61 y 0xa0b sum 0x8326a23
x 0x6769 y 0xc0d sum 0xd109678
x 0x6320 y 0xe0f sum 0x12822558
x 0x576f y 0x1011 sum 0x17fee3b7
x 0x7264 y 0x1213 sum 0x20126923
x 0x7320 y 0x1415 sum 0x291a5ac3
x 0x6172 y 0x1617 sum 0x3182e801
x 0x6520 y 0x1819 sum 0x3b07c821
x 0x5371 y 0x1a1b sum 0x438a0f0c
x 0x7565 y 0x1c1d sum 0x506e677d
```

```
 x 0x616d y 0x1e1f sum 0x5be4f9b0
 x 0x6973 y 0x2021 sum 0x6920f183
 x 0x6820 y 0x2223 sum 0x77036de3
 x 0x4f73 y 0x2425 sum 0x823b1582
 x 0x7369 y 0x2627 sum 0x936e4081
 x 0x6672 y 0x2829 sum 0xa38078c3
 x 0x6167 y 0x2a2b sum 0xb38bbb10
 x 0x6500 y 0x2c2d sum 0xc4f97c10
 sum mod F4, truncated to 16 bits: 0xb717
 pad variable: 0x708
 mmh16 value: 0xbe1f
MMH64 MAC
57 be cf 34 46 a9 be 1f
*/
```

Appendix III Certification Authority and Provisioning Guidelines (Informative)

III.1 Certificate Format and Extensions

This section describes the certificate format and extensions used by CableLabs certification authorities (CA) and summarizes the fields of [X.509] version 3 certificates. The CableLabs certificate PKI hierarchy is shown in Figure 22.

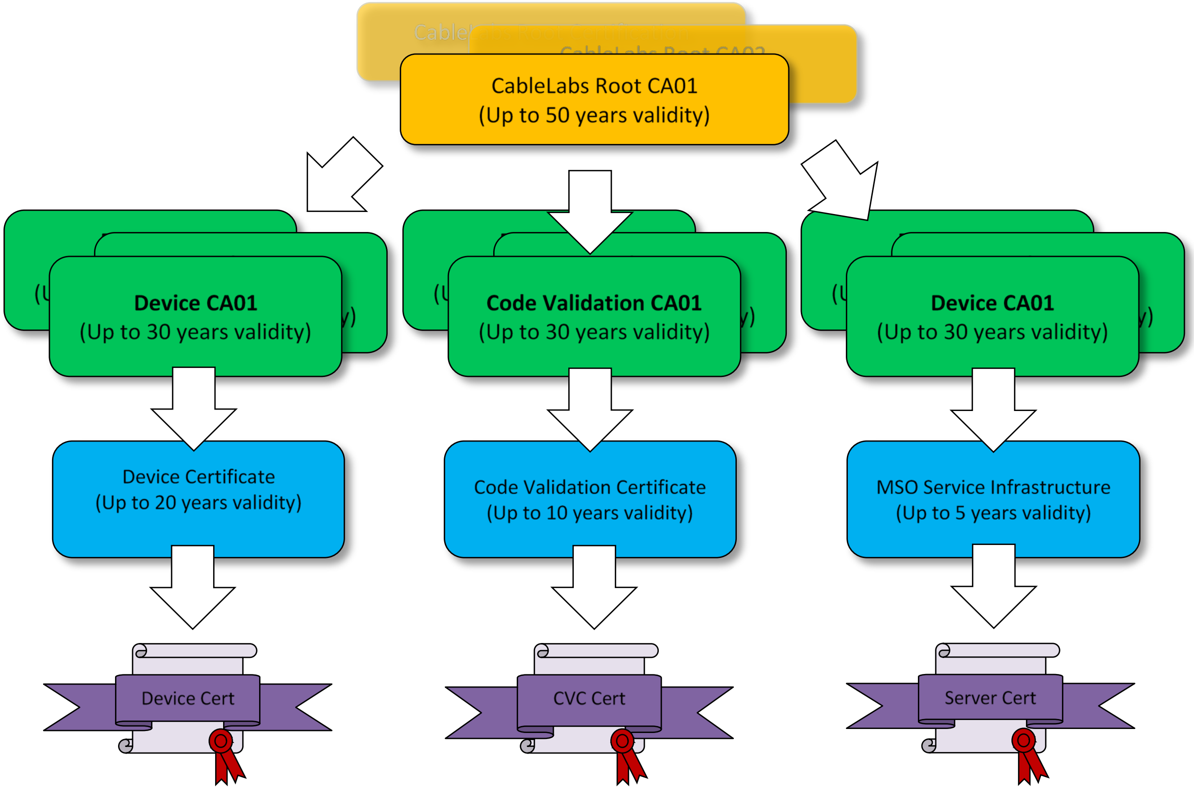
CableLabs Root CA01
(Up to 50 years validity)
Device CA01
(Up to 30 years validity)
Code Validation CA01
(Up to 30 years validity)
Device CA01
(Up to 30 years validity)
Device Certificate
(Up to 20 years validity)
Code Validation Certificate
(Up to 10 years validity)
MSO Service Infrastructure
(Up to 5 years validity)
Device Cert
CVC Cert
Server Cert

Figure 22 - Second Generation (2nd Gen) DOCSIS PKI Hierarchy ("New PKI")

All certificates and CRLs described in this specification are signed with the RSA signature algorithm, using SHA-256 as the hash function. The RSA signature algorithm is described in PKCS #1 [RSA1]; SHA-256 is described in [FIPS 180-4].

Names in [X.509] are SEQUENCEs of `RelativeDistinguishedNames`, which are in turn SETs of `AttributeTypeAndValue`. `AttributeTypeAndValue` is a SEQUENCE of an `AttributeType` (an OBJECT IDENTIFIER) and an `AttributeValue`. The value of the `countryName` attribute is a 2-character `PrintableString`, chosen from [ISO 3166]; all other `AttributeValues` are encoded as either `UTF8String` or `PrintableString` character strings. The `PrintableString` encoding is used if the character string contains only characters from the `PrintableString` set, specifically:

```
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789
'()+,-./:=? and space.
```

The `UTF8String` type is used if the character string contains characters not in the `PrintableString` set.

The DER-encoded `tbsCertificate.issuer` field of a valid DOCSIS certificate is an exact binary match to the DER-encoded `tbsCertificate.subject` field of its issuer certificate.

III.1.1 CableLabs OID Arc Management

The CableLabs OID (1.3.6.1.4.4491) is the base OID for the definition of identifiers used in CableLabs protocols. The CableLabs OID arc is organized as follows:

```
CableLabs OID ::= { 1.3.6.1.4.4491}
  |
  +-> id-cl-docsis-pki ::= {cl-id 2021}
      |
      +-> id-cl-docsis-pki-cp ::= {id-cl-docsis-pki 1}
      |
      +-> id-cl-docsis-pki-ext ::= {id-cl-docsis-pki 2}
          |
          +-> id-cl-docsis-pki-ext-eku ::= {id-cl-docsis-pki-ext 1}
```

Where values under the `id-cl-docsis-pki-ext` arc identify available functionality (e.g., CM or ONU). In order to request changes in the CableLabs OID arc, please contact the Policy Authority and follow the associated procedures.

III.1.1.1 Service OIDs for Extended Key Usage Values

The DOCSIS PKI defines several different values under the CableLabs DOCSIS PKI extensions arc `id-cl-docsis-pki-ext-eku (1.3.6.1.4.4491.2021.2.1)`. Table 62 provides the details of the different values and associated usage.

Table 62 - Object Identifiers for EKU-enabled Functionalities

Short Name	Name	Value	Description
svcCMTS	id-cl-pki-eku-CMTS	{id-cl-pki-ext-eku 1}	CMTS functionalities
svcCM	id-cl-pki-eku-CM	{id-cl-pki-ext-eku 2}	CM functionalities

III.1.1.2 Service OIDs Examples

The values in Table 62 are all relative to the `id-cl-pki-ext-eku` base OID. For example, the `svcCMTS (id-cl-pki-eku-CMTS)` and `svcCM (id-cl-pki-eku-CM)` that are used in CMTS and CM certificates, respectively, have the following dotted representation:

```
svcCMTS ::= { id-cl-pki-ext-eku 1 }
   --- Value: 1.3.6.1.4.4491.2021.2.1.1

svcCM ::= { id-cl-pki-ext-eku 2 }

   --- Value: 1.3.6.1.4.4491.2021.2.1.2
```

Refer to the Policy Authority for how to submit changes to this table.

III.2 CableLabs Root CA Certificate

The DOCSIS PKI comprises one or more Root Certification Authorities. Root Certification Authorities only issue Intermediate CA certificates (no EE certificates issued from the Root) and OCSP Responder ones. The profile for Root Certificates is defined in Table 63:

Table 63 - CableLabs Root CA Certificate

General Data	
Version	v3 (0x02)
Serial number	Unique Positive Integer assigned by the CA

Issuer DN	c=US
	o=CableLabs

	ou=Root CA<ID#>
	cn=CableLabs Root Certification Authority

Subject DN	c=US
	o=CableLabs
	ou=Root CA<ID#>
	cn=CableLabs Root Certification Authority

Validity Period	
Not Before	<Issuing Date>
Not After	<Issuing Date> + Up to 50 yrs
Public Key Info	
Public Key Algorithm	RSA (1.2.840.113549.1.1.1)
Signature Algorithm	Sha256WithRSAEncryption (1 2 840 113549 1 1 11) for RSA, or Sha384WithRSAEncryption (1 2 840 113549 1 1 11) for RSA, or Sha512WithRSAEncryption (1 2 840 113549 1 1 11) for RSA
Keysize	RSA:4096-bits or 8192-bits
Parameters	NULL

Extensions				
Standard Extensions	**OID**	**Required**	**Critical**	**Value**
keyUsage	{id-ce 15}	Yes	TRUE	
keyCertSign				Set (1)
cRLSign				Set (1)
digitalSignature				Set (1), or Not Set (0)
basicConstraints	{id-ce 19}	Yes	TRUE	
cA				Set (TRUE)
SubjectKeyIdentifier	{id-ce 14}	Yes	FALSE	
keyIdentifier				Set (<SHA-1 hash of the value of the BIT STRING subjectPublicKey (excluding the tag, length, and number of unused bits)>)
subjectAltName	{id-ce 17}	No	FALSE	(Deprecated)
directoryName				Set by the issuing CA

Values in angle brackets (<>) indicate that appropriate text as indicated below is present:

<ID#>: indicates the ID number of the CA and is populated with the CA certificate is issued (e.g., 01);

III.3 CableLabs Device CA Certificate

The CableLabs Device Certification Authority issues certificate for DOCSIS devices and it is issued by the Root CA only. For example, the Device CA is used to issue certificates for Cable Modems, CMTS, and Remote Phy Devices. The Device CA may also issue OCSP Responder certificates. The profile for the Device CA certificate is provided in Table 64:

Table 64 - CableLabs Device CA Certificate

General Data	
Version	v3 (0x02)
Serial number	Unique Positive Integer assigned by the CA

Issuer DN	c=US
	o=CableLabs
	ou=<RootCA Organization Unit>
	cn=<Root CA Name>

Subject DN	c=US
	o=CableLabs
	ou=Device CA<ID#>
	cn=CableLabs Device Certification Authority

Validity Period	
Not Before	<Issuing Date>
Not After	<Issuing Date> + Up to 30 yrs [*]
Public Key Info	
Public Key Algorithm	RSA (1 2 840 113549 1 1)
Signature Algorithm	Sha256WithRSAEncryption (1 2 840 113549 1 1 11) for RSA, or Sha384WithRSAEncryption (1 2 840 113549 1 1 12) for RSA, or Sha512WithRSAEncryption (1 2 840 113549 1 1 13) for RSA
Keysize	RSA: 3072-bits or 4096-bits
Parameters	NULL
Extensions	

Standard Extensions	**OID**	**Required**	**Critical**	**Value**
keyUsage	{id-ce 15}	**Yes**	**TRUE**	
keyCertSign				Set (1)
cRLSign				Set (1)
digitalSignature				Set (1), or Not Set (0)
basicConstraints	{id-ce 19}	**Yes**	**TRUE**	
cA				Set (TRUE)
pathLenConstraint				0
subjectKeyIdentifier	{id-ce 14}	**Yes**	FALSE	
keyIdentifier				Set (<SHA-1 hash of the value of the BIT STRING subjectPublicKey (excluding the tag, length, and number of unused bits)>)
authorityKeyIdentifier	{id-ce 35}	**Yes**	FALSE	
keyIdentifier				Set (<SHA-1 hash of the value of the BIT STRING subjectPublicKey (excluding the tag, length, and number of unused bits)>)
crlDistributionPoints	{id-ce 31}	No	FALSE	
distributionPoint				Set (<HTTP URI for Relevant CRL in DER format>)
certificatePolicies	{id-ce 32}	No	FALSE	
certPolicyId				Set (<DOCSIS PKI Certificate Policy OID>)
policyQualifiers				Not Set
authorityInfoAccess	{id-pe 1}	No	FALSE	
ocsp	{id-ad 1}			Set (<HTTP URI of the authoritative OCSP responder>)

caIssuers	{id-ad 2}			Set (<HTTP URI of the Issuing CA certificate in DER format>)
subjectAltName	{id-ce 17}	No	FALSE	**(Deprecated)**
directoryName				Set by the issuing CA for online CAs

[*] The certificate expiration cannot exceed the issuing CA's one.

III.4 DOCSIS 4.0 CM Certificate Profiles

Device Certificates are issued by Device Certification Authorities to DOCSIS certified cable modems. The profile for DOCSIS Device Certificate is provided in Table 65.

Table 65 - CableLabs DOCSIS 4.0 CM Certificate Profile

General Data				
Version	v3 (0x02)			
Serial number	Unique Positive Integer assigned by the CA			

Subject DN	c=<Country of Manufacturer>
	o=<Company Name>
	ou=<Manufacturing Location>
	cn=<Device Identifier>			

Validity Period				
Not Before	<Issuing Date>			
Not After	<Issuing Data> + Up to 20 yrs [*]			
Public Key Info				
Public Key Algorithm	RSA (1 2 840 113549 1 1)			
Signature Algorithm	Sha256WithRSAEncryption (1 2 840 113549 1 1 11) for RSA, or Sha384WithRSAEncryption (1 2 840 113549 1 1 12) for RSA, or Sha512WithRSAEncryption (1 2 840 113549 1 1 13) for RSA			
Keysize	RSA: 2048-bits, or 3072-bits, or 4096-bits			
Parameters	NULL			
Extensions				
Standard Extensions	**OID**	**Required**	**Critical**	**Value**
keyUsage	**{id-ce 15}**	**Yes**	**TRUE**	
digitalSignature				Set (1)
keyEncipherment				Set (1)
extendedKeyUsage	**{id-ce 37}**	**Yes**	**FALSE**	
svcCM				Set (id-cl-pki-ext-eku-CM)
clientAuth				Set (id-kp-clientAuth)
serverAuth				Set (id-kp-serverAuth)
authorityKeyIdentifier	**{id-ce 35}**	**Yes**	**FALSE**	
keyIdentifier				Set (<SHA-1 hash of the value of the BIT STRING subjectPublicKey (excluding the tag, length, and number of unused bits)>)
authorityInfoAccess	**{id-pe 1}**	**Yes**	**FALSE**	
ocsp	{id-ad 1}			Set (<HTTP URI of the authoritative OCSP responder>)
caIssuers	{id-ad 2}			Set (<HTTP URI of the Issuing CA certificate in DER format>), or Not Set
certificatePolicies	**{id-ce 32}**	**Yes**	**FALSE**	

certPolicyId				Set (<DOCSIS PKI Certificate Policy OID>)
policyQualifiers				Not Set
crlDistributionPoints	{id-ce 31}	No	FALSE	
distributionPoint				Set (<HTTP URI for Relevant CRL in DER format>)

[*] The expiration shall not exceed the issuing CA's one

Values in angle brackets (<>) indicate that appropriate text as indicated below is present:

<ID#>: indicates the ID number of the issuing CA (e.g., 01);

<Country of Manufacturer>: two-letter country code;

<Company Name>: name that identifies the company;

<Manufacturing Location>: name that identifies the location of manufacture;

<Device Identifier>: Device Identifier (e.g., MAC address of the CM).

CM Certificates use the device MAC Address as the <Device Identifier>.

The MAC Address is expressed as six pairs of hexadecimal digits separated by single colons (e.g., 00:60:21:A5:0A:23). Hexadecimal digits greater than 9 are expressed as uppercase letters.

The MAC address in the CM Certificate will be the same as the MAC address in the BPKM Attributes field.

CM Certificates include the `svcCM` value in the EKU.

Other non-critical extensions might be used in Device Certificates as needed.

III.5 DOCSIS 4.0 CMTS Certificate

Device Certificates are issued by **Device Certification Authorities** to DOCSIS 4.0 Cable Modem Termination Systems or CMTS.

The profile for DOCSIS 4.0 CMTS Certificates is provided in Table 66.

Table 66 - CableLabs DOCSIS 4.0 CMTS Certificate Profile

General Data	
Version	v3 (0x02)
Serial number	Unique Positive Integer assigned by the CA

Subject DN	c=<Country of Manufacturer>
	o=<Company Name>
	ou=<Manufacturing Location>
	cn=<Device Identifier>

Validity Period	
Not Before	<Issuing Date>
Not After	<Issuing Date> + Up to 5 years [*]
Public Key Info	
Public Key Algorithm	RSA (1 2 840 113549 1 1)
Signature Algorithm	Sha256WithRSAEncryption (1 2 840 113549 1 1 11) for RSA, or Sha384WithRSAEncryption (1 2 840 113549 1 1 12) for RSA, or Sha512WithRSAEncryption (1 2 840 113549 1 1 13) for RSA
Key size	RSA: 2048-bits, or 3072-bits, or 4096-bits
Parameters	NULL

Extensions				
Standard Extensions	**OID**	**Required**	**Critical**	**Value**
keyUsage	{id-ce 15}	**Yes**	**TRUE**	
digitalSignature				Set (1)
keyEncipherment				Set (1)
extendedKeyUsage	{id-ce 37}	**Yes**	FALSE	
svcCMTS				Set (id-cl-pki-ext-eku-CMTS)
clientAuth				Set (id-kp-clientAuth)
serverAuth				Set (id-kp-serverAuth)
authorityKeyIdentifier	{id-ce 35}	**Yes**	FALSE	
keyIdentifier				Set (<SHA-1 hash of the value of the BIT STRING subjectPublicKey (excluding the tag, length, and number of unused bits)>)
authorityInfoAccess	**{id-pe 1}**	**Yes**	**FALSE**	
ocsp	{id-ad 1}			Set (<HTTP URI of the authoritative OCSP responder>)
caIssuers	{id-ad 2}			Set (<HTTP URI of the Issuing CA certificate in DER format>)
authorityInfoAccess	**{id-pe 1}**	**Yes**	**FALSE**	
ocsp	{id-ad 1}			Set (<HTTP URI of the authoritative OCSP responder>)
caIssuers	{id-ad 2}			Set (<HTTP URI of the Issuing CA certificate in DER format>) or Not Set
certificatePolicies	**{id-ce 32}**	**Yes**	**FALSE**	
certPolicyId				Set (<DOCSIS PKI Certificate Policy OID>)
policyQualifiers				Not Set
crlDistributionPoints	{id-ce 31}	No	FALSE	
distributionPoint				Set (<HTTP URI for Relevant CRL in DER format>)
subjectAltName	{id-ce 17}	No	FALSE	
dNSName				Set (<FQDN>), or Not Set

[*] The expiration shall not exceed the issuing CA's one.

Values in angle brackets (<>) indicate that appropriate text as indicated below is present:

<ID#>: indicates the ID number of the issuing CA (e.g., 01);

<Country of Manufacturer>: two-letter country code;

<Company Name>: name that identifies the company;

<Manufacturing Location>: name that identifies the location of manufacture;

<Device Identifier>: Meaningful identifier for the device (e.g., FQDN or Device MAC address).

When a MAC Address is used for the <Device Identifier>, the value of the MAC Address is expressed as six pairs of hexadecimal digits separated by single colons (e.g., 00:60:21:A5:0A:23). Hexadecimal digits greater than 9 are expressed as uppercase letters.

III.6 CableLabs DOCSIS CVC CA Certificate

The DOCSIS CVC CA is used to issue certificates for Code Validation and it is issued by the Root CA only. This type of certificates is used for authenticating Software Images (e.g., for Secure Software Download). CVC CAs may also issue certificates for OCSP Responders. The profile for CVC CA certificates is provided in Table 67:

Table 67 - CableLabs DOCSIS CVC CA Certificate

CableLabs CVC CA Certificate Profile	
Version	v3 (0x02)
Serial number	Unique Positive Integer assigned by the CA

Issuer DN	c=US
	o=CableLabs
	ou=<Root CA Organization Unit>
	cn=<Root CA Name>

Subject DN	c=US
	o=CableLabs
	ou=CVC CA<ID#>
	cn=CableLabs CVC Certification Authority

Validity Period	
Not Before	<Issuing Date>
Not After	<Issuing Date> + Up to 30 yrs [*]
Public Key Info	
Public Key Algorithm	RSA (1 2 840 113549 1 1)
Signature Algorithm	Sha256WithRSAEncryption (1 2 840 113549 1 1 11) for RSA, or Sha384WithRSAEncryption (1 2 840 113549 1 1 12) for RSA, or Sha512WithRSAEncryption (1 2 840 113549 1 1 13) for RSA
Keysize	RSA: 3072-bits, or 4096-bits
Parameters	NULL
Extensions	

Standard Extensions	OID	Required	Critical	Value
keyUsage	{id-ce 15}	Yes	**TRUE**	
keyCertSign				Set (1)
cRLSign				Set (1)
digitalSignature				Set (1), or Not Set (0)
basicConstraints	{id-ce 19}	**Yes**	**TRUE**	
cA				Set (TRUE)
pathLenConstraint				Set (0)
subjectKeyIdentifier	{id-ce 14}	**Yes**	FALSE	
keyIdentifier				Set (<SHA-1 hash of the value of the BIT STRING subjectPublicKey (excluding the tag, length, and number of unused bits)>)
authorityKeyIdentifier	{id-ce 35}	**Yes**	FALSE	
keyIdentifier				Set (<SHA-1 hash of the value of the BIT STRING subjectPublicKey (excluding the tag, length, and number of unused bits)>)
crlDistributionPoints	{id-ce 31}	No	FALSE	
distributionPoint				Set (<HTTP URI for Relevant CRL in DER format>)
certificatePolicies	{id-ce 32}	No	FALSE	
certPolicyId				Set (<DOCSIS PKI Certificate Policy OID>)

policyQualifiers				Not Set
authorityInfoAccess	{id-pe 1}	No	FALSE	
ocsp	{id-ad 1}			Set (<HTTP URI of the authoritative OCSP responder>)
caIssuers	{id-ad 2}			Set (<HTTP URI of the Issuing CA certificate in DER format>)
subjectAltName	{id-ce 17}	No	FALSE	**(Deprecated)**
directoryName				Set by the issuing CA for online CAs

[*] The expiration cannot exceed the issuing CA's one

Values in angle brackets (<>) indicate that appropriate text as indicated below is present:

<Root CA Organization Unit>: OU value copied from the issuing CA

<Root CA Name>: CN copied from the issuing Root CA

<ID#>: indicates the ID number of the CA and is populated when the CA certificate is issued (e.g., 01)

<Country of Manufacturer>: two-letter country code

<Company Name>: name that identifies the company

III.7 Code Verification Certificate

Code Verification Certificates (or CVCs) are used to authenticate software images. For example, this type of certificate is used to sign Firmware images that are then loaded onto Cable Modems (or other devices like RPD) via the Secure Software Download. The details about the Code Verification Certificate profile are provided in Table 68:

Table 68 - Code Verification Certificate

<table>
<tr><th colspan="5">General Data</th></tr>
<tr><td colspan="2"><b>Version</b></td><td colspan="3">v3 (0x02)</td></tr>
<tr><td colspan="2"><b>Serial number</b></td><td colspan="3">Unique Positive Integer assigned by the CA</td></tr>
<tr><td colspan="2"><b>Issuer DN</b></td><td colspan="3">c=US
o=CableLabs
ou=CVC CA<ID#>
cn=CableLabs CVC Certification Authority</td></tr>
<tr><td colspan="2"><b>Subject DN</b></td><td colspan="3">c=<Country of Manufacturer>
o=<Company Name>
cn=Code Verification Certificate</td></tr>
<tr><th colspan="5">Validity Period</th></tr>
<tr><td colspan="2"><b>Not Before</b></td><td colspan="3"><Issuing Date></td></tr>
<tr><td colspan="2"><b>Not After</b></td><td colspan="3"><Issuing Date> + Up to 10 yrs [*]</td></tr>
<tr><th colspan="5">Public Key Info</th></tr>
<tr><td colspan="2"><b>Public Key Algorithm</b></td><td colspan="3">RSA (1 2 840 113549 1 1)</td></tr>
<tr><td colspan="2"><b>Signature Algorithm</b></td><td colspan="3">Sha256WithRSAEncryption (1 2 840 113549 1 1 11) or,
Sha384WithRSAEncryption (1.2.840.113549.1.1.12) or,
Sha512WithRSAEncryption (1.2.840.113549.1.1.13)</td></tr>
<tr><td colspan="2"><b>Keysize</b></td><td colspan="3">RSA: 2048-bits, 3072-bits, or 4096-bits</td></tr>
<tr><td colspan="2"><b>Parameters</b></td><td colspan="3">NULL</td></tr>
<tr><th colspan="5">Extensions</th></tr>
<tr><th>Standard Extensions</th><th>OID</th><th>Required</th><th>Critical</th><th>Value</th></tr>
<tr><td><b>extendedKeyUsage</b></td><td>{id-ce 37}</td><td><b>Yes</b></td><td><b>TRUE</b></td><td></td></tr>
</table>

codesigning				Set (id-kp-codeSigning)
keyUsage	{id-ce 15}	No	**TRUE**	
digitalSignature				Set (1), or Not Set (0)
authorityKeyIdentifier	{id-ce 35}	**Yes**	FALSE	
keyIdentifier				Set (<SHA-1 hash of the value of the BIT STRING subjectPublicKey (excluding the tag, length, and number of unused bits)>)
crlDistributionPoints	{id-ce 31}	No	FALSE	
distributionPoint				Set (<HTTP URI for Relevant CRL in DER format>)
certificatePolicies	{id-ce 32}	No	FALSE	
certPolicyId				Set (<DOCSIS PKI Certificate Policy OID>)
policyQualifiers				Not Set
authorityInfoAccess	{id-pe 1}	No	FALSE	
ocsp	{id-ad 1}			Set (<HTTP URI of the authoritative OCSP responder>)
caIssuers	{id-ad 2}			Set (<HTTP URI of the Issuing CA certificate in DER format>)
subjectAltName	{id-ce 17}	No	FALSE	**(Deprecated)**
directoryName				Set by the issuing CA for online CAs

[*] The expiration cannot exceed the issuing CA's one

Values in angle brackets (<>) indicate that appropriate text as indicated below is present:

<Country of Manufacturer>: two-letter country code;

<Company Name>: name that identifies the company

Co-signer CVCs will have a unique numeric value for the <Company Name> which is assigned by CableLabs. The value is a printable string of eight hexadecimal digits. Each hexadecimal digit in the name is chosen from the ranges 0x30 to 0x39 or 0x41 to 0x46. The string 0x3030303030303030 is not assigned.

III.8 CableLabs DOCSIS Service Provider CA Certificate

Service Provider CAs are used to issue certificates for the operator's infrastructure, and it is issued by the Root CA only. For example, Service Provider CAs issue certificates for AAA servers, TLS servers, etc. The profile for Service Provider CA Certificates is provided in Table 69:

Table 69 - CableLabs DOCSIS Service Provider CA Certificate

General Data	
Version	v3 (0x02)
Serial number	Unique Positive Integer assigned by the CA

Issuer DN	c=US
	o=CableLabs
	ou=<Root CA Organization Unit>
	cn=<Root CA Name>

Subject DN	c=US
	o=CableLabs
	ou=Service Provider CA<ID#>
	cn=CableLabs Service Provider Certification Authority

Validity Period	
Not Before	<Issuing Date>
Not After	<Issuing Date> + Up to 30 years [*]
Public Key Info	

Public Key Algorithm	RSA (1 2 840 113549 1 1)			
Signature Algorithm	Sha256WithRSAEncryption (1 2 840 113549 1 1 11) or, Sha384WithRSAEncryption (1.2.840.113549.1.1.12) or, Sha512WithRSAEncryption (1.2.840.113549.1.1.13)			
Keysize	RSA: 3072-bits or 4096-bits			
Parameters	NULL			
Extensions				
Standard Extensions	**OID**	**Required**	**Critical**	**Value**
keyUsage	{id-ce 15}	**Yes**	**TRUE**	
keyCertSign				Set (1)
cRLSign				Set (1)
digitalSignature				Set (1), or Not Set (0)
basicConstraints	{id-ce 19}	**Yes**	**TRUE**	
cA				Set (TRUE)
pathLenConstraint				Set (0)
subjectKeyIdentifier	{id-ce 14}	**Yes**	FALSE	
keyIdentifier				Set (<SHA-1 hash of the value of the BIT STRING subjectPublicKey (excluding the tag, length, and number of unused bits)>)
authorityKeyIdentifier	{id-ce 35}	**Yes**	FALSE	
keyIdentifier				Set (<SHA-1 hash of the value of the BIT STRING subjectPublicKey (excluding the tag, length, and number of unused bits)>)
crlDistributionPoints	{id-ce 31}	No	FALSE	
distributionPoint				Set (<HTTP URI for Relevant CRL in DER format>)
certificatePolicies	{id-ce 32}	No	FALSE	
certPolicyId				Set (<DOCSIS PKI Certificate Policy OID>)
policyQualifiers				Not Set
authorityInfoAccess	{id-pe 1}	No	FALSE	
ocsp	{id-ad 1}			Set (<HTTP URI of the authoritative OCSP responder>)
caIssuers	{id-ad 2}			Set (<HTTP URI of the Issuing CA certificate in DER format>)
subjectAltName	{id-ce 17}	No	FALSE	
directoryName				**(Deprecated)** Set by the issuing CA for online CAs

[*] The expiration cannot exceed the issuing CA's one

Values in angle brackets (<>) indicate that appropriate text as indicated below is present:

<Root CA Organization Unit>: OU value copied from the issuing CA

<Root CA Name>: CN copied from the issuing Root CA

<ID#>: indicates the ID number of the CA and is populated when the CA certificate is issued (e.g., 01)

III.9 CableLabs DOCSIS Service Provider Certificate

Service Provider Certificates are used to authenticate the operator's DOCSIS infrastructure (e.g., AAA servers, TLS servers, etc.). The DOCSIS credentials can be validated by any entity (e.g., a Cable Modem, a CCAP Core, an RMD, etc.) that is participating in the trust infrastructure. The profile for Service Provider Certificates is provided in Table 70:

Table 70 - CableLabs Service Provider Certificate

<table>
<tr><th colspan="5">General Certificate Data</th></tr>
<tr><td colspan="2">Version</td><td colspan="3">v3 (0x02)</td></tr>
<tr><td colspan="2">Serial number</td><td colspan="3">Unique Positive Integer assigned by the CA</td></tr>
<tr><td colspan="2">Issuer DN</td><td colspan="3">c=US
o=CableLabs
ou=Service Provider CA<ID#>
cn=CableLabs Service Provider Certification Authority</td></tr>
<tr><td colspan="2">Subject DN</td><td colspan="3">c=<Country Code>
o=<Company Name>
ou=Service Provider Certificate
cn=<Common Name></td></tr>
<tr><th colspan="5">Validity Period</th></tr>
<tr><td colspan="2">Not Before</td><td colspan="3"><Issuing Date></td></tr>
<tr><td colspan="2">Not After</td><td colspan="3"><Issuing Date> + Up to 5 yrs [*]</td></tr>
<tr><th colspan="5">Public Key Info</th></tr>
<tr><td colspan="2">Public Key Algorithm</td><td colspan="3">RSA (1 2 840 113549 1 1)</td></tr>
<tr><td colspan="2">Signature Algorithm</td><td colspan="3">Sha256WithRSAEncryption (1 2 840 113549 1 1 11) or,
Sha384WithRSAEncryption (1.2.840.113549.1.1.12) or,
Sha512WithRSAEncryption (1.2.840.113549.1.1.13)</td></tr>
<tr><td colspan="2">Keysize</td><td colspan="3">RSA:2048-bits, or 3072-bits, or 4096-bits</td></tr>
<tr><td colspan="2">Parameters</td><td colspan="3">NULL</td></tr>
<tr><th colspan="5">Extensions</th></tr>
<tr><th>Standard Extensions</th><th>OID</th><th>Required</th><th>Critical</th><th>Value</th></tr>
<tr><td>keyUsage</td><td>{id-ce 15}</td><td><b>Yes</b></td><td><b>TRUE</b></td><td></td></tr>
<tr><td>digitalSignature</td><td></td><td></td><td></td><td>Set (1)</td></tr>
<tr><td>keyEncipherment</td><td></td><td></td><td></td><td>Set (1)</td></tr>
<tr><td>extendedKeyUsage</td><td>{id-ce 37}</td><td><b>Yes</b></td><td><b>TRUE</b></td><td></td></tr>
<tr><td>enabledFunction [**]</td><td></td><td></td><td></td><td>Set (<Enabled Function>), or Not Set</td></tr>
<tr><td>serverAuth</td><td></td><td></td><td></td><td>Set ({id-kp-serverAuth}), or Not Set</td></tr>
<tr><td>clientAuth</td><td></td><td></td><td></td><td>Set ({id-kp-clientAuth}), or Not Set</td></tr>
<tr><td>authorityKeyIdentifier</td><td>{id-ce 35}</td><td><b>Yes</b></td><td>FALSE</td><td></td></tr>
<tr><td>keyIdentifier</td><td></td><td></td><td></td><td>Set (<SHA-1 hash of the value of the BIT STRING subjectPublicKey (excluding the tag, length, and number of unused bits)>)</td></tr>
<tr><td>subjectAltName</td><td>{id-ce 17}</td><td>No</td><td>FALSE</td><td></td></tr>
<tr><td>dNSName</td><td></td><td></td><td></td><td>Set (<Server’s FQDN>), or Not Set</td></tr>
<tr><td>otherName
nai_on_realm</td><td>{1.3.6.1.5.5.7.8.8}</td><td></td><td></td><td>Set (<Server’s Realm>), or Not Set</td></tr>
<tr><td><b>crlDistributionPoints</b></td><td>{id-ce 31}</td><td>No</td><td>FALSE</td><td></td></tr>
<tr><td>distributionPoint</td><td></td><td></td><td></td><td>Set (<HTTP URI for Relevant CRL in DER format>)</td></tr>
<tr><td><b>certificatePolicies</b></td><td>{id-ce 32}</td><td>No</td><td>FALSE</td><td></td></tr>
<tr><td>certPolicyId</td><td></td><td></td><td></td><td>Set (<DOCSIS PKI Certificate Policy OID>)</td></tr>
<tr><td>policyQualifiers</td><td></td><td></td><td></td><td>Not Set</td></tr>
<tr><td><b>authorityInfoAccess</b></td><td>{id-pe 1}</td><td>No</td><td>FALSE</td><td></td></tr>
<tr><td>ocsp</td><td>{id-ad 1}</td><td></td><td></td><td>Set (<HTTP URI of the authoritative OCSP responder>)</td></tr>
</table>

General Certificate Data				
caIssuers	{id-ad 2}			Set (<HTTP URI of the Issuing CA certificate in DER format>)

[*] The expiration cannot exceed the issuing CA's one

Values in angle brackets (<>) indicate that appropriate text as indicated below is present:

<ID#>: indicates the ID number of the issuing CA (e.g., 01)

<Country Code>: two-letter country code

<Company Name>: name that identifies the company

<Common Name>: meaningful name or identifier for the service

Other non-critical extensions might be used in Service Provider certificates as requested by operators.

III.10 Certificate Installation

The following certificates are installed in the CMTS by either the supplier during manufacturing or cable operator during installation:

- CableLabs Root CA Certificate (used for validating CM certificates and firmware upgrades.
- The Issuing CableLabs Device CA Certificate (used to chain to the CMTS to the CableLabs Root CA Certificate).
- CMTS Device Certificate together with its corresponding private key.

The following certificates are installed in the CM during manufacturing:

- CableLabs Root CA Certificate (used for validating code download images).
- The Issuing CableLabs Device CA Certificate (used to chain the CM to the CableLabs Root CA certificate).
- CM Device Certificate and its corresponding private key.

III.11 CM Code File Signing Policy and Format

CM vendors and cable operators can control the Secure Software Download process based on their policies by updating the Manufacturer/Co-Signer CVC or by changing the signingTime in the Manufacturer/Co-Signer CVS (Code Verification Signature). At this time, the DOCSIS 4.0 specifications do not specify the policy related to the CM Code File signing process. However, an example of the policy is specified in this section.

III.11.1 Manufacturer CM Code File Signing Policy

In order to sign code files, the manufacturer obtains a valid CVC from the DOCSIS Root CA.

When signing a code file, a manufacturer may choose not to update the FWAH (i.e., [PKCS#7]) `signingTime` value in the manufacturer signing information. The FWAH signingTime value should be equal to or greater than the CVC's validity start time. The FWAH `signingTime` value is not less than the CVC's validity start time.

The CM vendor and its Manufacturer Code Signing Agent (Mfg CSA), which securely stores the RSA private key corresponding to the RSA public key in the Manufacturer CVC and generates the CVS for the CM Code File, might employ the following policy for the CM Code File signing process.

The Mfg CSA continues to put exactly the same date and time value (T1) in the signingTime field in the Mfg CVS of the CM Code File as long as the vendor does not have any CM Code File to revoke.

Once the vendor realizes there are certain issues in one or more CM Code File(s) and wants to revoke them, the vendor chooses the current date and time value (T2) and starts using T2 as the signingTime value in the Mfg CVS for all the newly created CM Code File from that point. In addition, any CM Code files signed with T1 that are still good are re-signed using T2.

Under this policy, because the multiple CM Code Files make a group of the CM Code Files with the exact same signingTime value in the Msg CVS, the operator can download any CM Code File in the group in any order. That is, among the CM Code Files in the same group, the CM's software can be downgraded if necessary.

III.11.2 Operator CM Code File Signing Policy

Operators should verify that the code image as received from a vendor is exactly as built by the trusted manufacturer.

The operator may choose to co-sign code images destined for use on its network.

All code images downloaded to a CM across the network are signed in accordance with this specification.

III.11.3 CM Code File Format

A single file is used to encapsulate the code for the cable modem. The CM code file is composed of a Firmware Authentication Header (FWAH) and a `SignedContent` field.

The signed data message includes:

- The Manufacturer Code Verification Signature (CVS);
- The Manufacturer Code Verification Certificate (CVC);
- The issuing CVC CA Certificate;
- The code image (compatible with the destination CM) as signed content;
- Optionally, when the MSO co-signs the code file:
  a) The Co-Signer CVS;
  b) The Co-signer CVC;
  c) The issuing CVC CA Certificate; this certificate does not need to be separately added when it is identical to the issuing CVC CA certificate of the Manufacturer CVC.
- Optional Device CA Certificate
- Optional Root CA Certificate

The code file complies with [PKCS#7] and is DER encoded. The code file matches the structure shown below. A code file example is shown in Annex D.

Table 71 - CM Code File

Code File Structure	Description
[PKCS#7] Digital Signature{	
ContentInfo	
contentType	SignedData
signedData()	EXPLICIT signed-data content value; includes CVS and [X.509] CVC
}	
SignedContent{	
DownloadParameters {	Mandatory TLV format (Type 28) defined in the Section 7.2.2.28. (Length is zero if there are no sub-TLVs.)
DeviceCACert()	Optional TLV for one DER-encoded new PKI certificate formatted according to the Device-CA-Certificate TLV format (Type 53) defined in the Section 7.2.2.31.
RootCACert()	Optional TLV for one DER-encoded new PKI certificate formatted according to the Root-CA-Certificate TLV format (Type 54) defined in the Section 7.2.2.32.
}	
CodeImage()	Upgrade code image

Code File Structure	Description
}	

III.11.3.1 DOCSIS PKCS#7 Signed Data

The signedData field of the DOCSIS [PKCS#7] Digital Signature matches the DER encoded structure defined in Appendix III.

III.11.3.1.1Code Signing Keys

The digital signature uses the RSA Encryption Algorithm [RSA3] with SHA-256 [FIPS 180-4]. The RSA key modulus is at least 2048 bits in length.

Table 72 - DOCSIS PKCS#7 Signed Data

PKCS#7 Field	Description
signedData {	
Version	version = 1
digestAlgorithmIdentifiers	SHA-256
contentInfo	
contentType	data (SignedContent is concatenated at the end of the [PKCS#7] structure)
certificates {	DOCSIS Code Verification Certification (CVC)
mfgCVC()	Required for all code files
mfgCVCCA()	Required for all code files using the new PKI
cosignerCVC()	Optional; required for cable operator co-signatures
cosignerCVCCA()	Optional; required for cable operator co-signatures using the new PKI when cosignerCVCCA certificate is not identical to the mfgCVCCA certificate.
} *end certificates*	
SignerInfo{	
MfgSignerInfo {	Required for all code files
Version	version = 1
issuerAndSerialNumber	from the signer's certificate
issuerName	distinguished name of the certificate issuer
certificateSerialNumber	from CVC; Integer, size (1..20) octets
digestAlgorithm	SHA-256
authenticatedAttributes	
contentType	data; contentType of signedContent
signingTime	UTCTime (GMT), YYMMDDhhmmssZ
messageDigest	digest of the content as defined in [PKCS#7]
digestEncryptionAlgorithm	rsaEncryption
encryptedDigest	
} *end mfg signer info*	
MsoSignerInfo {	OPTIONAL; required for cable operator co-signatures
Version	version =1
issuerAndSerialNumber	from the signer's certificate
issuerName	distinguished name of the certificate issuer
certificateSerialNumber	from CVC; Integer, size (1..20) octets
digestAlgorithm	SHA-256
authenticatedAttributes	
contentType	data; contentType of signedContent
signingTime	UTCTime (GMT), YYMMDDhhmmssZ

PKCS#7 Field	Description
messageDigest	digest of the content as defined in [PKCS#7]
digestEncryptionAlgorithm	rsaEncryption
encryptedDigest	
} *end mso signer info*	
} *end signer info*	
} *end signed data*	

III.11.3.1.2 Code Verification Certificate Format

The format used for the CVC is defined in Appendix III.7.

III.11.3.1.3 Code Verification Certificate Revocation

In addition to standard certificate revocation, there is a method for revoking CVCs based on the validity start date of the certificate. This method requires that an updated CVC be delivered to the cable modem with an updated validity start time. Once the CVC is successfully validated, the validity start time will update the CM's current value of cvcAccessStart (see Section 14.3.2.2).

To expedite the delivery of an updated CVC without requiring the cable modem to process a code upgrade, the CVC may be delivered in either the CM's configuration file or an SNMP MIB. The format of a DOCSIS CVC is the same whether it is in a code file, configuration file, or SNMP MIB.

III.11.3.2 Signed Content

The SignedContent field of the code file contains the CodeImage and the DownloadParameters fields, which may contain up to two items: a Root CA Certificate and a Device CA Certificate.

The final code image is in a binary format compatible with the destination CM. In support of the [PKCS#7] signature requirements, the code content is encoded as an OCTET STRING.

Each manufacturer should build their code with additional mechanisms to verify that an upgrade code image is compatible with the destination CM.

Appendix IV Acknowledgements (Informative)

On behalf of the cable industry and our member companies, CableLabs would like to thank the following individuals for their contributions to the development of this specification.

Contributor	Company Affiliation
Margo Dolas	Broadcom
Massimiliano Pala, Craig Pratt, Stuart Hoggan, Yuan Tian, Volker Leisse, Doug Jones	CableLabs
Philip Anderson	Charter
Pawel Sowinski	Cisco
Ramy Elmoneiry	Comcast
Sasha Medvinsky, Jeff DeMent, Ali Negahdar, Dan Torbet	CommScope
Jeff Finkelstein, David Taylor, Owen Parsons,	Cox Communications
Satish Mudugere, Onur Zengin	MaxLinear
Colin Dearborn	Shaw Communications

Since this specification is based on the [DOCSIS BPI+] specification the following individuals and organizations who participated in its development are also acknowledged.

Contributor	Company Affiliation
Mike St. Johns	@Home Corporation
Burcak Beser	3com
Kirk Erichsen	Adelphia
Doc Evans	ARRIS Interactive
Bill Aiello	AT&T
Mohan Gundu	BigBand Networks
Doug Nguyen, Jeff Carr, Lisa Denney, Niki Pantelias	Broadcom Corporation
Matt Schmitt, Oscar Marcia, Stuart Hoggan	CableLabs
Man Wong, Shengyou Zeng	Cisco Systems
James Yee, John Pickens, Mike Sabin	Com21
Brian Epstein, Rick Mayberry	Comcast
Rusty Cashman	Correlant Communications
Ben Bekele	Cox Cable
Han-Seung Koo	ETRI
Jonathan Fellows, Sasha Medvinsky	General Instrument Corporation
Lior Storfer	Libit Signal Processing Ltd.
Katherine Unge, Mark Sumner, Mike Patrick, Nathan Vanderschaaf	Motorola Corporation
Stuart Green, Wilson Sawyer	Nortel Networks
George Hart	Rogers Cable TV Ltd.
Lucy Pollak	Texas Instruments
Kazuyoshi Ozawa	Toshiba

Appendix V Revision History

The following Engineering Changes were incorporated into CM-SP-SECv4.0-I02-201202:

ECN Identifier	Accepted Date	Title of EC	Author
SECv4.0-N-20.2073-2	02/13/2020	CMTS certificate processing	Jones
SECv4.0-N-20.2098-1	05/21/2020	CM SSH updates	Jones
SECv4.0-N-20.2113-3	09/17/2020	Addition of BPI+ V2	Jones

The following Engineering Changes were incorporated into CM-SP-SECv4.0-I03-210826:

ECN Identifier	Accepted Date	Title of EC	Author
SECv4.0-N-20.2138-1	12/10/2020	Assigning TLV 98 for SECv4.0 CM SSH Server Configuration Settings (Annex A.1.2)	Jones
SECv4.0-N-21.2157-2	4/29/2021	Code Verification Certificate (CVC) updates	Jones
SECv4.0-N-21.2159-1	4/29/2021	Updates for Device Certificate Profiles and Extended Key Usage (EKU) checks	Jones
SECv4.0-N-21.2173-2	6/24/2021	Guidelines for a CM to authenticate to a cloud server	Torbet
SECv4.0-N-21.2175-1	7/1/2021	Adjusting TLV 98 sub-TLVs	Jones
SECv4.0-N-21.2176-1	7/1/2021	Adding reference to TLV 99 to SECv4.0	Jones

The following Engineering Change was incorporated into CM-SP-SECv4.0-I04-220328:

ECN Identifier	Accepted Date	Title of EC	Author
SECv4.0-N-22.2237-1	3/17/2022	Update SECv4.0 for version 5 BPKM messages plus other cleanup	Tian

✻ ✻ ✻