Data-Over-Cable Service Interface Specifications
DOCSIS 3.0

Security Specification

CM-SP-SECv3.0-I04-070518

ISSUED

Notice

Document Status Sheet

Document Control Number:	CM-SP-SECv3.0-I04-070518			
Document Title:	Security Specification			
Revision History:	I01 – Released August 4, 2006 I02 – Released December 22, 2006 I03 – Released February 23, 2007 I04 – Released May 18, 2007			
Date:	May 18, 2007			
Status:	~~Work in Progress~~	~~Draft~~	Issued	~~Closed~~
Distribution Restrictions:	~~DOCSIS 3.0 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 stable document, which has undergone rigorous member and vendor review and is suitable for product design and development, cross-vendor interoperability, and for certification testing.

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 third generation of high-speed data-over-cable systems. This specification was developed for the benefit of the cable industry, and includes contributions by operators and vendors from North America, Europe, and other regions.

1.2 Background

1.2.1 Broadband Access Network

A coaxial-based broadband access network is assumed. This may take the form of either an all-coax or 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 in each direction, although typical maximum separation may be 10-15 miles.
- A maximum differential optical/electrical spacing between the CMTS and the closest and most distant modems of 100 miles in each direction, although this would typically be limited to 15 miles.

At a propagation velocity in fiber of approximately 1.5 ns/ft, 100 miles of fiber in each direction results in a round-trip delay of approximately 1.6 ms.

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 1-1.

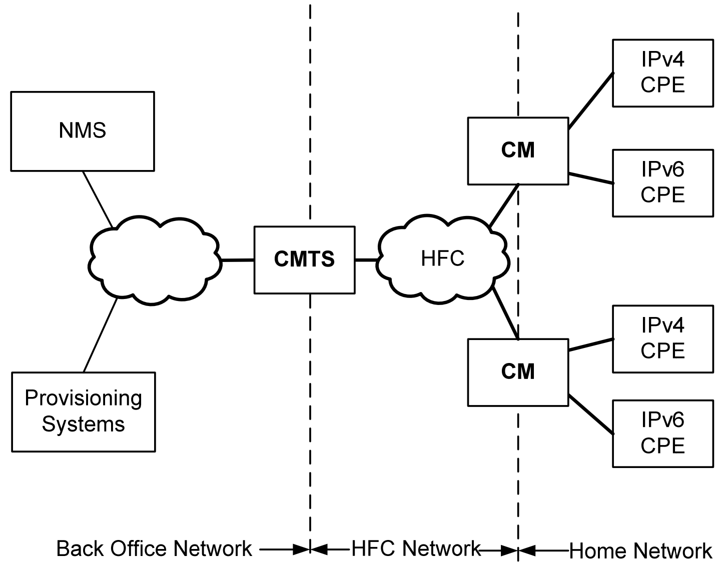

Figure 1-1 - The DOCSIS Network

The CM connects to the operator's cable network and to a home network, bridging packets between them. Many CPE devices can connect to the CM's 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 1-1). CPE devices may use IPv4, IPv6, or both forms of IP addressing. Examples of typical CPE devices are home routers, set-top devices, personal computers, etc.

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 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. Applications include:

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 Time Protocol server provides Time Protocol clients, typically CMs, with the current time of day.
- Certificate Revocation server provides certificate status.

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 IPv4 address space limitations have become a constraint on network operations. To this end, CableLabs' member companies have decided to add new features to the DOCSIS® specification for the purpose of increasing channel capacity, enhancing network security, expanding addressability of network elements, and deploying new service offerings.

The DOCSIS system allows transparent bi-directional transfer of Internet Protocol (IP) traffic, between the cable system head-end and customer locations, over an all-coaxial or hybrid-fiber/coax (HFC) cable network. This is shown in simplified form in Figure 1-2.

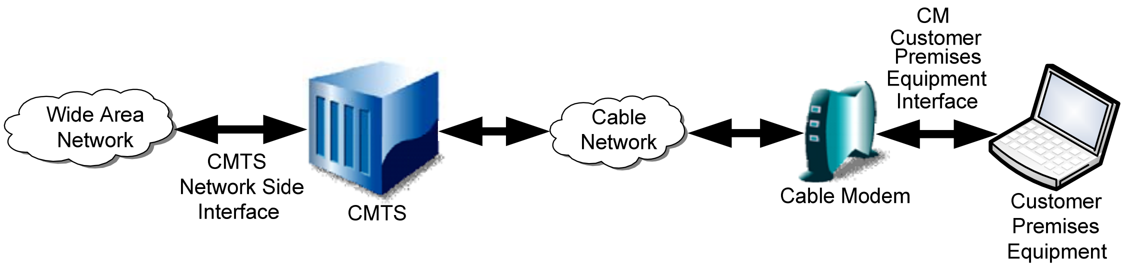

Figure 1-2 - Transparent IP Traffic Through the Data-Over-Cable System

1.2.4 Statement of Compatibility

This specification defines the DOCSIS 3.0 interface. Prior generations of DOCSIS were commonly referred to as DOCSIS 1.0, 1.1 and 2.0. DOCSIS 3.0 is backward-compatible with equipment built to the previous specifications. DOCSIS 3.0-compliant CMs interoperate seamlessly with DOCSIS 2.0, DOCSIS 1.1 and DOCSIS 1.0 CMTSs. DOCSIS 3.0-compliant CMTSs seamlessly support DOCSIS 2.0, DOCSIS 1.1 and DOCSIS 1.0 CMs.

Refer to Annex D for BPI/BPI+ compatibility requirements.[1]

[1] This sentence added per SECv3.0-N-07.0409-1, #6 on 4/27/07 by KN.

1.2.5 Reference Architecture

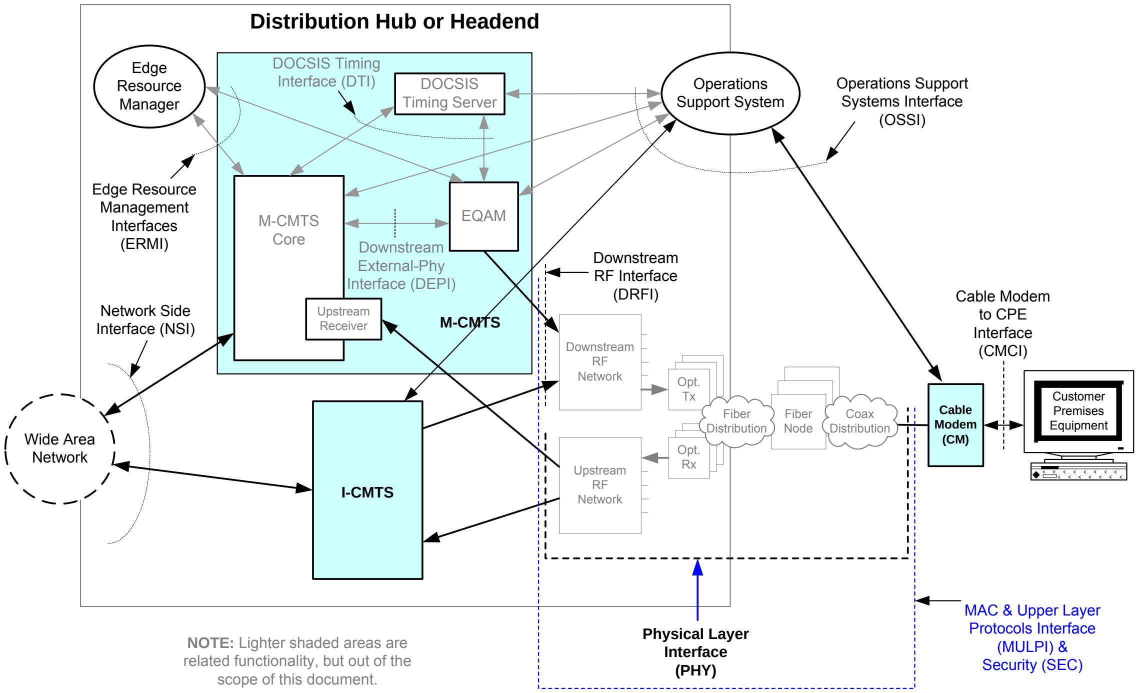

Figure 1-3 - Data-over-Cable Reference Architecture

The reference architecture for data-over-cable services and interfaces is shown in Figure 1-3. The lighter shaded areas are related functionality, but are out of the scope of DOCSIS specifications. Boxes represent functional components and arrows represent interfaces.

1.2.6 DOCSIS 3.0 Documents

A list of the specifications in the DOCSIS 3.0 series is provided in Table 1-1. For further information, please refer to http://www.cablemodem.com.

Table 1-1 - DOCSIS 3.0 Series of Specifications

Designation	Title
CM-SP-PHYv3.0	Physical Layer Specification
CM-SP-MULPIv3.0	Media Access Control and Upper Layer Protocols Interface Specification
CM-SP-OSSIv3.0	Operations Support System Interface Specification
CM-SP-SECv3.0	Security Specification

This specification defines security requirements.

A list of related DOCSIS specifications are listed in Table 1-2.

Table 1-2 - DOCSIS 3.0 Related Specifications

Designation	Title
CM-SP-eDOCSIS	eDOCSIS™ Specification
CM-SP-CMCI	Cable Modem CPE Interface 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-ERMI	Edge Resource Manager Interface Specification
CM-SP-M-OSSI	M-CMTS Operations Support System Interface Specification
CM-SP-L2VPN	Layer 2 Virtual Private Networks 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 must 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.

[DOCSIS BPI+] Data-Over-Cable Service Interface Specifications, Baseline Privacy Interface Specification, CM-SP-BPI+-I12-050812, August 12, 2005, Cable Television Laboratories, Inc.

[DOCSIS MULPI 3.0] DOCSIS-over-Cable Service Interface Specifications, Media Access Control and Upper Layer Protocols Interface Specification, CM-SP-MULPIv3.0-I04-070518, May 18, 2007, Cable Television Laboratories, Inc.

[DOCSIS OSSI 3.0] DOCSIS-over-Cable Service Interface Specifications, Operations Support System Interface Specification, CM-SP-OSSIv3.0-I03-070518, May 18, 2007, Cable Television Laboratories, Inc.

[DOCSIS RFI 2.0] Data-Over-Cable Service Interface Specifications, Radio Frequency Interface Specification, CM-SP-RFIv2.0-I11-060602, June 2, 2006, Cable Television Laboratories, Inc.

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

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

[FIPS-180-2] Federal Information Processing Standards Publication (FIPS PUB) 180-2, Secure Hash Standard, February 2003.

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

[ISO8859-1] ISO, 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-I12-050812, August 12, 2005, Cable Television Laboratories, Inc.

[RFC 826] IETF RFC 826, D.C. Plummer, Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48.bit Ethernet address for transmission on Ethernet hardware, Nov-01-1982.

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

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

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

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

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

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

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

[RFC 2461] IETF RFC 2461, T.Narten, E. Nordmark, W. Simpson, Neighbor Discovery for IP Version 6 (IPv6), December 1998.

[RFC 2560] IETF RFC 2560, M.Myers *et al*., “X.509 Internet Public Key Infrastructure Certificate Status Protocol – OCSP”, June 1999.

[RFC 3280] IETF RFC 3280, R. Housley, W. Ford, W. Polk, D. Solo, “Internet X.509 Public Key Infrastructure Certificate and CRL Profile”, April 2002.

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

[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 22-2] ANSI/SCTE 22-2 2002, DOCSIS 1.0 Baseline Privacy Interface.

[SCTE 52] ANSI/SCTE 52 2003, Data Encryption Standard Cipher Block Chaining Pocket Encryption.

[X.509] ITU-T Recommendation X.509 (1997): Information Technology - Open Systems Interconnection - The Directory: Authentication Framework, August 1997.

[X.690] ITU-T Recommendation X.690 (2002) | 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 CMCI] Data-Over-Cable Service Interface Specifications Cable Modem to Customer Premise Equipment Interface Specification SP-CMCI-I10-050408, April 8, 2005, Cable Television Laboratories, Inc.

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

[RFC 1750] IETF RFC 1750, D. Eastlake, et al., “Randomness Recommendations for Security”, December 1994.

[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, H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, RTP: A Transport Protocol for Real-Time Applications, July 2003.

[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.

[X.680] ITU-T Recommendation X.680, (July, 2002): Abstract Syntax Notation One (ASN.1): Specification of basic notation.

2.3 Reference Acquisition

American National Standards Institute, Inc. 1819 L Street, NW, 6th floor Washington, DC 20036, Phone +1-202-293-8020; Fax +1-202-293-9287. http://www.ansi.org.

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.

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/.

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/.

3 TERMS AND DEFINITIONS

This specification uses the following terms:

DER Encoded	Refers to 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.
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.
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.
Static Multicast Sessions	Multicast sessions joined during cable modem registration.
Upstream	The term used to describe traffic and paths that go from the subscriber to the headend.

4 ABBREVIATIONS AND ACRONYMS

This specification uses the following abbreviations and acronyms:[2]

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
CFB	Cipher Feedback
CM	Cable Modem
CMCI	Cable Modem to Customer Premises Equipment Interface
CMTS	Cable Modem Termination System
CMTS-NSI	Cable Modem Termination System Network Side Interface
CPE	Consumer Premises Equipment
CRC	Cyclic Redundancy Check
CRL	Certificate Revocation List
CRT	Chinese Remainder Theorem
CTR	Counter: the counter mode of a block cipher
CVC	Code Verification Certificate
CVS	Code Verification Signature
DA	Destination Address
DER	Distinguished Encoding Rules
DES	Data Encryption Standard
DHCP	Dynamic Host Configuration Protocol
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
EDE	Encrypt-Decrypt-Encrypt
EH	Extended Header
EHDR	Extended MAC Header

[2] Revised this list per ECN SECv3.0-N-06.0330-1 by GO on 1/15/07.

F_n	The nth Fermat number
FC	Frame Control
FCRC	Fragment Cyclic Redundancy Check
FIPS	Federal Information Processing Standard
FSM	Finite State Machine
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	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
MSB	Most Significant Bit
MSO	Multiple Systems Operator
NMS	Network Management System
OCSP	Online Certificate Status Protocol
OID	Object Identifier
OSS	Operations Support System
OUI	Organizationally Unique Identifier
PDU	Protocol Data Unit
PHS	Payload Header Suppression
PKI	Public Key Infrastructure
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
SET	Secure Electronic Transaction
SHA-1	Secure Hash Algorithm 1
SID	Service Identifier
SNMP	Simple Network Management Protocol
SSD	Secure Software Download
TEK	Traffic Encryption Key
TFTP	Trivial File Transfer Protocol
TLV	Type/Length/Value
ToD	Time of Day
UTC	Coordinated Universal Time
XOR	Exclusive Or

5 OVERVIEW

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

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

This specification provides operators with tools to secure the provisioning process of Cable Modems (CM) and protect Cable Modem 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 QoS;
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 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+], which in turn was based on the original DOCSIS Baseline Privacy scheme [SCTE 22-2].

This specification explicitly assumes the following to be true:

- It is acceptable that DOCSIS 3.0 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 3.0 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 New DOCSIS 3.0 Security Features

DOCSIS 3.0 introduces a number of features that build on previous versions of DOCSIS. This specification includes the following new features for security:

- 128 bit AES traffic encryption;
- Early CM authentication and traffic encryption (EAE);

- Enhanced secure provisioning features:
- Source IP address verification (SAV);
- TFTP proxy and configuration file learning;
- MMH algorithm for CMTS MIC:
- Certificate Revocation;
- Update support of the CableLabs CVC Root CA and CableLabs CVC CA certificates for future use in secure software download; and
- Encryption support for new method of multicast messaging.[3]

5.2 Technical Overview

This specification defines the Base Line Privacy Plus (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).

5.2.1 BPI+ Architecture

BPI+ has two component protocols:

- An encapsulation protocol for encrypting packet data across the cable network. This protocol defines:
    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;
    3. The rules for applying those algorithms to a DOCSIS MAC frame's packet data.
- A key management protocol (Baseline Privacy Key Management, BPKM) to provide 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.

5.2.1.1 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 element within the MAC Extended Header (see [DOCSIS MULPI 3.0]).

DOCSIS 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.

5.2.1.2 Key Management Protocol

CMs use the Baseline Privacy Key Management (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.

[3] Bullet added per SECv3.0-N-07.0409.1, #5 on 4/26/07 by KN.

The BPKM protocol adheres to a client/server model, where 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 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 Early Authentication and Encryption[4] (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.

5.2.1.3 DOCSIS Security Associations

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

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

A Security Association's shared information comprises the cryptographic suite, traffic encryption keys and CBC initialization vectors, and the lifetime of the keying information. Each Security Association 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 hence, 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).

[4] Changes to paragraph 2 in Section 5.2.1.2 made by KN on 11/17/06 per EC SECv3.0-N-06.0322-1.

A CMTS MAY support Static SAs.

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

An SA's keying material (*e.g.,* 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 CM and CMTS maintain key synchronization.

5.2.1.4 QoS SIDs and DOCSIS SAIDs

The BPI+ Extended Header Element in downstream DOCSIS MAC frames ([DOCSIS MULPI 3.0]) 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, 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, 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 a SAID in their extended headers; instead, the Baseline Privacy EH element may contain a valid QoS SID assigned to the CM (see Section 6.3 for details).

The Baseline Privacy Extended Header 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, it provides a mechanism for issuing piggybacked bandwidth requests (see Section 6.3.1); it can also carry fragmentation control data (see Section 0). These two functions are tied to a particular QoS SID; for this reason, upstream Baseline Privacy Extended Header 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 a SAID is being used as a Primary SA, 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 a SAID to a Static or Dynamic SA, 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 a MD-DS-SG.

5.2.2 Secure Provisioning

The processes used to provision a CM are: DHCP, ToD, and TFTP at the IP layer; and registration at the MAC layer. Securing these provisioning processes play 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 & 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 TFTP server and validates the new software image's digital signature before installation.

5.3 Operation

5.3.1 Cable Modem Initialization

The [DOCSIS MULPI 3.0] 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);
- CMTS Registration.

BPI+ security may be established between the CM and the CMTS following CM ranging (Early Authentication and Encryption (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.[5]

BPI+ security initialization begins with the CM sending the CMTS an Authentication Information message, containing the CM's Manufacturer CA Certificate, and an Authorization Request message. The Authorization Request contains the following information:

- Data identifying the CM (*e.g.,* MAC address and other information unique to the device);
- The CM's RSA public key;
- A digital certificate to bind the CM's identifying data to the CM's public key[6]:
- A list of the cryptographic suites supported by the CM;
- The Initialization SAID (value 0).

If the CMTS successfully authenticates the requesting CM, it responds with an Authorization Reply, containing the Primary SAID for the CM, and an Authorization Key encrypted by the public key of the CM's certificate. 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

[5] Changes made to paragraph per SECv3.0-N-07.0379.1, #1 on 4/26/07 by KN.
[6] This means that the public key appears twice in the message, once as a standalone entity and once as part of the certificate.

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.

5.3.1.1 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, it does not allow it 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 MULPI 3.0] specification.

5.3.1.2 EAE and Authentication Reuse

When EAE is enabled, CM authentication normally occurs following ranging completion, 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 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 restrict other security features such as configuration file learning (see Section 9.4.2.4).

5.3.1.3 Configuration Registration Enforcement

To help prevent theft-of-service attacks caused by alterations to the configuration file, 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 for them 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 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. This helps hide the TFTP server address from exposure on the cable network where it might be captured and used for a 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.[7]

5.3.2 Cable Modem Key Update Mechanism

The Traffic Encryption Keys (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 that they are using match those that 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.

[7] Changes to paragraph 2 in Section 5.3.1.3 made by KN on 11/17/06 per EC SECv3.0-N-06.0322-1.

In addition, CMs are required to periodically reauthorize with the CMTS (see Section 7.1.4). 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.

5.3.3 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 TFTP 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 SNMP commands.

After a CM downloads a software image, it validates the image by verifying that the included CVC chains to the DOCSIS Root CA, and by checking the image's digital signature. If this validation is successful, 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 of 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 Extended Header Element in the DOCSIS MAC Header identifies the Security Association and accompanying keying material that is used to encrypt the PDU. The CM MUST NOT encrypt (see Section 6.5) MAC Management messages, except REG-REQ-MP messages, unless they are part of a fragment.

6.2 CMTS Requirements[8]

When communicating with a CM for which DOCSIS security is enabled, the CMTS MUST encrypt the Protocol Data Unit (PDU) regions of variable-length PDU MAC Frames and variable-length Isolation PDU MAC Frames [DOCSIS MULPI 3.0].

A Baseline Privacy Extended Header Element in the DOCSIS MAC Header identifies the Security Association and accompanying keying material that is used to encrypt the PDU. The CMTS MUST NOT encrypt (see Section 6.5) MAC Management messages.

6.3 Variable-Length PDU MAC Frame Format[9]

Figure 6-1 depicts the format of a DOCSIS variable-length PDU MAC Frame or variable-length Isolation PDU MAC Frame, with a Privacy Extended Header (EH) Element and encrypted PDU payload.

[8] Revised this section per ECN SECv3.0-N-06.0296-1 by GO on 10/13/06.
[9] Revised this section per ECN SECv3.0-N-06.0296-1 by GO on 10/13/06.

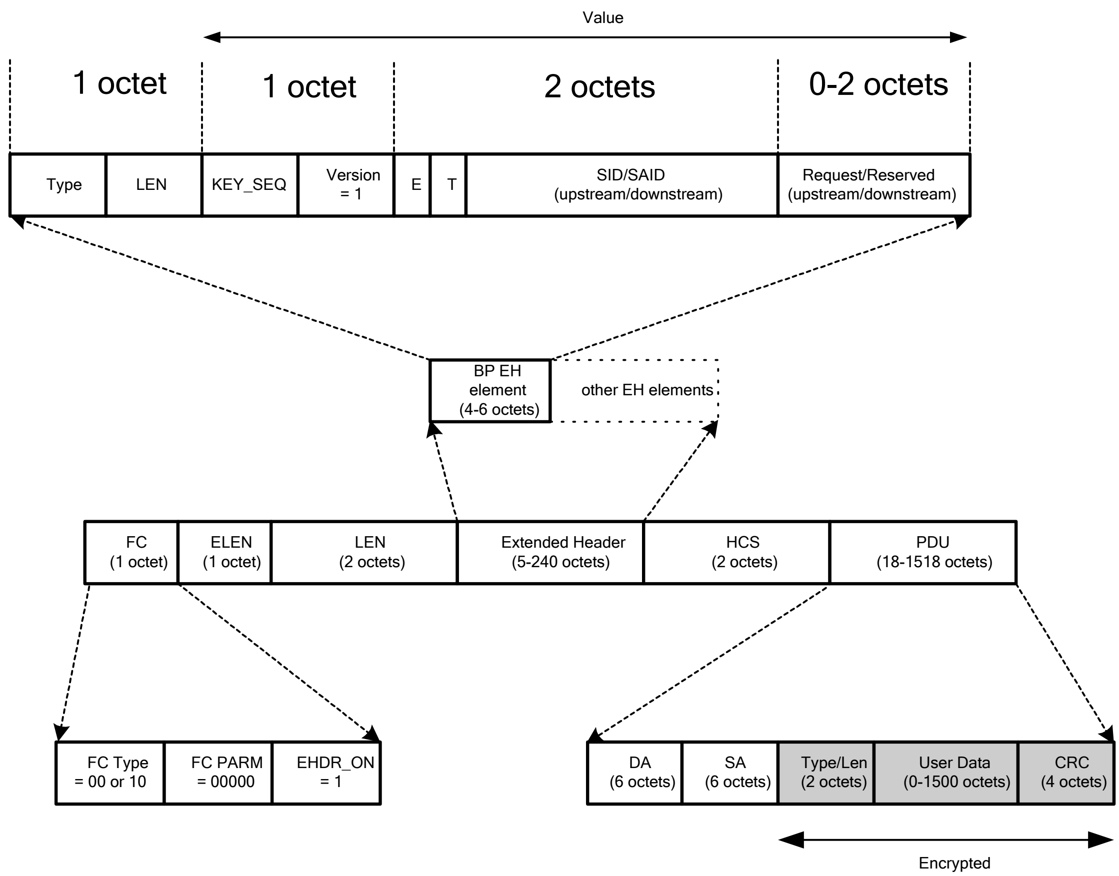

Figure 6-1 - Format of DOCSIS Variable-length PDU with Privacy EH Element

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 CRC. The CM MUST encrypt the PDU's Ethernet/802.3 CRC.

The CM MUST include a Baseline Privacy Extended Header in all frames containing encrypted PDUs. The CM MUST make the Baseline Privacy Extended Header element the first Extended Header in an upstream frame. Upstream frames sent by the CM MUST contain the `Type` value BP_UP or BP_UP2 [DOCSIS MULPI 3.0] in the Baseline Privacy EH.

The CMTS MUST include a Baseline Privacy Extended Header in all frames containing encrypted PDUs. The Baseline Privacy Extended Header 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 MULPI 3.0] in the Baseline Privacy EH. The high-order bits of a Baseline Privacy Extended Header `Value` field contains 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 SID/SAID (SID for upstream frames, SAID for downstream frames).

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

The DOCSIS MAC protocol [DOCSIS MULPI 3.0] defines a Request EH element that is used to piggyback a bandwidth request on an upstream data transmission. The last octet or last two octets of the upstream Baseline Privacy EH carries an optional piggybacked bandwidth allocation request.

In downstream packets, the last octet is reserved [DOCSIS MULPI 3.0] and the CMTS MUST set its value to zero.

Table 6–1 - Summary of the contents of Baseline Privacy Extended Headers

TYPE	LENGTH	VALUE
BP_UP See [DOCSIS MULPI 3.0]	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 mini-slots requested for upstream bandwidth.
BP_UP2 See [DOCSIS MULPI 3.0]	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.

TYPE	LENGTH	VALUE
BP_UP2 See [DOCSIS MULPI 3.0]	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.
BP_DOWN See [DOCSIS MULPI 3.0]	4	KEY_SEQ (4 bits), Version (4 bits), SID (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 MULPI 3.0]. When functioning in this role, the upstream Baseline Privacy Extended Header is extended by one octet, the additional octet serving as a fragmentation control field. Figure 6-2 depicts the format of a DOCSIS Fragmentation MAC Frame with an encrypted fragmentation payload.

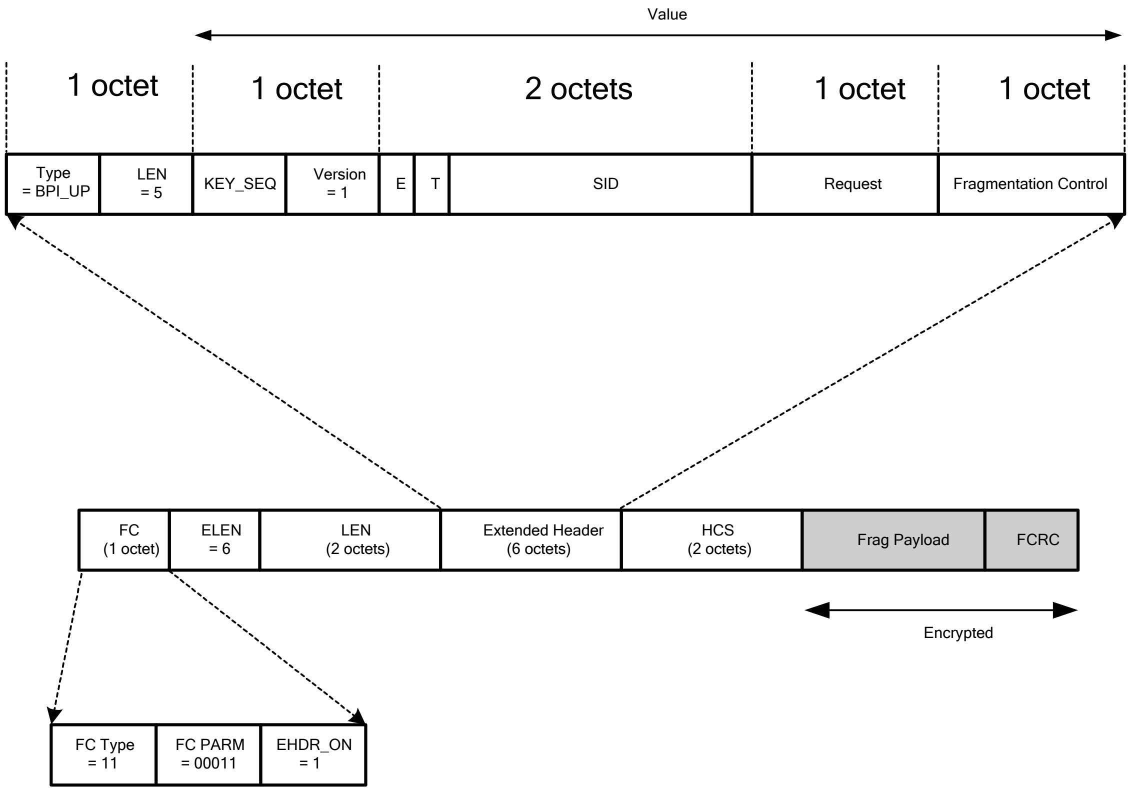

Figure 6-2 - Format of a DOCSIS MAC Fragmentation Frame with an Encrypted Payload

Frames with the fragmentation 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 6-2 - Summary of the contents of a DOCSIS Fragmentation MAC Frame's Baseline Privacy Extended Header

TYPE	LENGTH	VALUE
BP_UP See [DOCSIS MULPI 3.0]	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 mini-slots requested for upstream bandwidth. Fragmentation Control field contains fragmentation-specific control information; see [DOCSIS MULPI 3.0] for details.

The CM determines whether a packet will be fragmented based on its knowledge of the grant size (*i.e.,* the number of mini-slots a CMTS grants to a CM in an Upstream Bandwidth Allocation MAP [DOCSIS MULPI 3.0]). 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). Figure 6-1 depicts the format of a DOCSIS MAC management message frame with a Privacy Extended Header (EH) Element and encrypted payload.

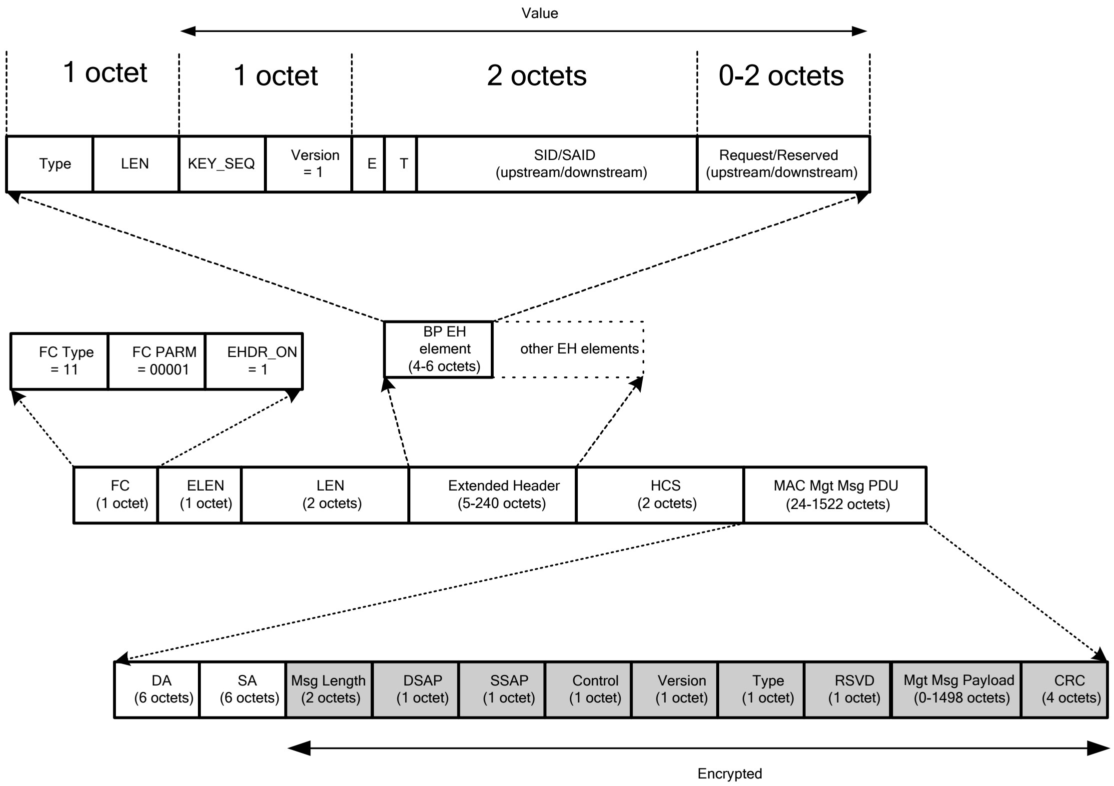

Figure 6-3 - Format of a DOCSIS MAC Management Message Frame with Encrypted Payload

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 Extended Header MUST be included in the frame;
- The Baseline Privacy Extended Header 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, which 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 6-3 - Summary of the contents of DOCSIS MAC Management Message Baseline Privacy Extended Headers

TYPE	LENGTH	VALUE
BP_UP See [DOCSIS MULPI 3.0]	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 mini-slots requested for upstream bandwidth.
BP_UP2 See [DOCSIS MULPI 3.0]	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 MULPI 3.0]	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), the CMTS SHOULD NOT place a BP Extended Header element in the frame.

If encryption is not enabled for a CM's unicast traffic, the CM MUST include the Baseline Privacy Extended Header 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, the CM MUST disable encryption on that frame. If Baseline Privacy EHDR is present on such a frame, 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, the CMTS MUST disable encryption on that frame. If an Baseline Privacy EHDR is present on such a frame, 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: an Authorization state machine and a TEK state machine. This section describes these two state machines. 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 behave identically to the state machines described in this section. The external behavior of CM implementations MUST behave identically to the state machines described in this section.

CM authorization, controlled by the Authorization state machine, is the process of:

- the CMTS authenticating a client CM's identity;
- the CMTS providing the authenticated CM with an Authorization Key, from which a Key Encryption Key (KEK) and message authentication keys are derived;
- the CMTS providing the authenticated CM with the identities (*i.e.,* the SAIDs) and properties of primary and static security associations for which the CM is authorized to obtain keying information.

The KEK is a two-key, 3DES encryption key that the CMTS uses to encrypt Traffic Encryption Keys (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 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 Manufacturer CA or CableLabs Mfg CA certificate, issued by the DOCSIS Root CA. 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 CMTS to learn these CA certificates from its client CMs. See Section 7.1.4.2.5 for more details.

Immediately after sending the Authentication Information message, 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 Security Associations in which the CM is authorized to participate. The Authorization Request contains:

- The CM's manufacturer ID and serial number;
- The CM's MAC address;
- The CM's public key;
- A CM Device certificate binding the CM's public key to its other identifying information[10];

[10] This means that the public key appears twice in the message, once as a standalone entity and once as part of the certificate.

- A description of the cryptographic algorithms supported by the CM. 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 7.2.1.1). The CM MUST use a value of zero for the Initialization SAID. The CMTS MUST interpret a SAID value of zero as an Initialization SAID.

In response to an Authorization Request message, the CMTS:

- Verifies the CM Device certificate;
- Checks that the CMTS and the CM share at least one cryptographic suite;
- Assigns the CM's Primary SAID.

In addition, the CMTS MUST verify that the MAC-address attribute of the Auth Request, the MAC address in the CM's Device certificate, and the source MAC address of the Auth Request MAC message, all match.

Furthermore, 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:

1. create an Authorization Key for the CM;
2. encrypt the Authorization Key with the CM's public key; and
3. send the encrypted Authorization Key to the CM in an Authorization Reply message.

The Authorization Reply contains:

- An Authorization Key encrypted with the CM's public key;
- A four (4) bit key sequence number;
- A key lifetime;
- The identities (*i.e.,* the SAIDs) and properties of the Primary and zero or more Static Security Associations for which the CM is authorized to obtain keying information.

If the CMTS supports Static SAs, the CMTS MUST include in the Authorization Reply 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) in the Authorization Reply. Upon receiving an Authorization Reply, 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). Dynamic SAs are not included in the Authorization Reply and therefore are not shown in Figure 7-1.

7.1.1.2 TEK State Machine Overview

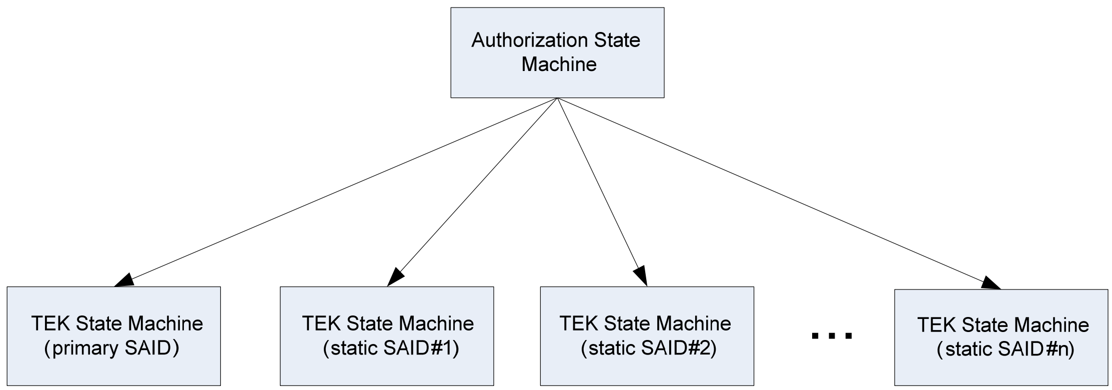

Figure 7-1 - Relationship Among Authorization and TEK State Machines

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, consisting of the manufacturer ID, serial number, MAC address and RSA Public Key;
- the SAID whose keying material is being requested;
- an HMAC keyed message digest, authenticating the Key Request.

The CMTS checks the HMAC digest of the Key Request message (see Section 11 and 12). If the CMTS verifies the HMAC, 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 Traffic Encryption Key (TEK);
- A Cipher Block Chaining (CBC) initialization vector;
- The key sequence number;
- The key's remaining lifetime;
- An HMAC keyed message digest.

The TEK in the Key Reply is 3DES (EDE mode) encrypted, using a two-key, 3DES Key Encryption Key (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 a 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 a 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 7.1.4).

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 7-1), 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[11]

The message exchange between the CMTS and the CM for the signaling and initialization of multicast traffic encryption is dependant 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 (IGMP for IPv4 and MLD for IPv6) message. Such multicast sessions are called Dynamically-joined Multicast Sessions. The cable operator can configure the cable modem to join multicast sessions during registration. Such multicast sessions are called Static Multicast Sessions [DOCSIS MULPI 3.0].

A CM indicates support for DSID Multicast Forwarding in the Registration Request message with a Multicast DSID Forwarding capability encoding [DOCSIS MULPI 3.0];. A CM that reports in this encoding the value of either 1 (GMAC-Explicit) or 2 (GMAC-Promiscuous) is said to support Multicast DSID forwarding. 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

[11] Modified this section per ECN SECv3.0-N-06.0345-2 by GO on 1/15/07.

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 MULPI 3.0]; such a CM is said to be an MDF disabled CM.[12]

Security Associations 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 MULPI 3.0]). 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 Security Associations.

7.1.2.1 Signaling of Dynamic and Static Multicast Session SAs when MDF is Disabled[13]

If a CM does not support Multicast DSID Forwarding, 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. For backwards compatibility, the CM MUST support dynamic SA-MAP messaging as described in Annex C.

If the CMTS disables MDF for a CM that supports MDF, 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, 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. If the CMTS disables MDF, the CM MUST signal SAs using the DOCSIS 1.1/2.0 Dynamic Security Association mechanism described in Annex C.

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 MULPI 3.0], 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. This 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 for purposes other than multicast encryption to an MDF-enabled CM. The CM accepts static SAs in BPI-Auth Reply messages, even when it operates in MDF enabled mode.

[12] This paragraph modified per SECv3.0-N-07.0409-1, # 12 on 4/27/07 by KN.
[13] Changes to this section made per SECv3.0-N-07.0409, #8 on 4/27/07 by KN.

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, 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, 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, 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, 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, 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, the CM MUST reject the REG-RSP(-MP) message.

7.1.3 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. A 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. A 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.

7.1.4 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 7-2) and as a state transition matrix (Table 7-1).

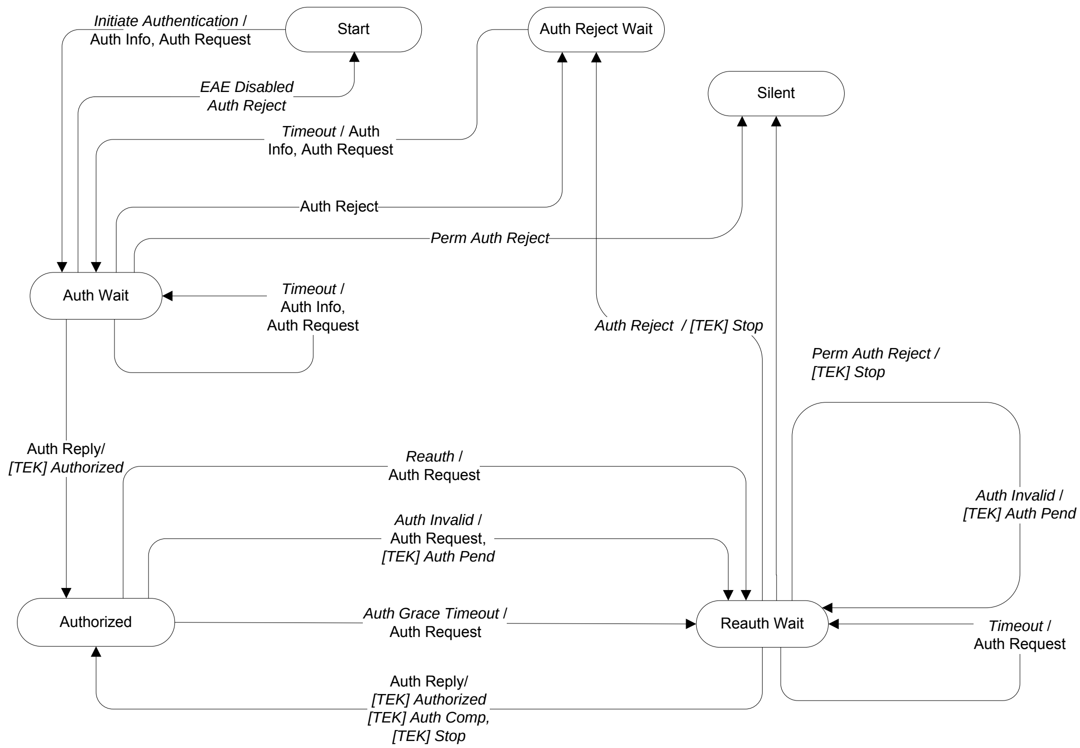

Figure 7-2 - Authorization State Machine Flow Diagram

Table 7-1 - 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]			

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 7-3:

- 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 7-1, 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 7.1.4.7.

A shaded cell within the state transition matrix (Table 7-1), 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.

7.1.4.1 Brief Description of States

7.1.4.1.1 [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.

7.1.4.1.2 [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.

7.1.4.1.3 [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.

7.1.4.1.4 [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.

7.1.4.1.5 [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 transitioned to the [Auth Reject Wait] state. The CM remains in this state until the timer expires.

7.1.4.1.6 [Silent]

The CM received an Authorization Reject message in response to its last Authorization Request. The Authorization Reject's error code indicated that the error was permanent. This triggers a transition to the [Silent] state.

In the [Silent] state, the CM;

- MUST NOT pass CPE traffic; and
- If the CM has a valid IP address, it MUST respond to SNMP management requests arriving from the cable network.

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

7.1.4.2 Brief Description of Messages

The corresponding message formats are specified in Section 7.2.

7.1.4.2.1 Authorization Request (Auth Request)

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

7.1.4.2.2 Authorization Reply (Auth Reply)

Receive an Authorization Key and list containing the primary SAID and static SAIDs. The Authorization Reply is sent from CMTS to CM. The Authorization Key is encrypted with the CM's public key.

7.1.4.2.3 Authorization Reject (Auth Reject)

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

7.1.4.2.4 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 re-authorize 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.

7.1.4.2.5 Authentication Information (Auth Info)

The Authentication Information message contains the Manufacturer CA or CableLabs Mfg CA certificate, issued by the DOCSIS Root CA. The Auth Info message is a message sent by the CM to the CMTS. The CMTS MUST first use the out-of-band configuration information to obtain the Manufacturer CA or CableLabs Mfg CA certificate (see Section 13.3.2). If the CMTS does not learn the Manufacturer CA or CableLabs Mfg CA certificate from the out-of-band configuration information then the CMTS MUST use the Manufacturer CA or CableLabs Mfg CA certificate from the Auth Info message sent by the CM.

7.1.4.3 Brief Description of Events

7.1.4.3.1 Initiate Authentication }

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

7.1.4.3.2 {Timeout}

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

7.1.4.3.3 {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 re-authorize 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.

7.1.4.3.4 {Reauth}

This event is generated in response to an SNMP SET ([DOCSIS OSSI 3.0]) that is intended to trigger a reauthorization cycle because the CM's set of authorized static SAIDs may have changed.

7.1.4.3.5 {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.

7.1.4.3.6 {Perm Auth Reject}

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

7.1.4.3.7 {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.

7.1.4.3.8 {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.

7.1.4.4 Events sent to TEK State Machine

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

7.1.4.4.1 {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.

7.1.4.4.2 {TEK Authorized}

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

7.1.4.4.3 {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.

7.1.4.4.4 {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 a {Auth Pend} event.

7.1.4.5 Brief Description of Timing Parameters

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

7.1.4.5.1 Authorize Wait Timeout (Auth Wait Timeout)

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

7.1.4.5.2 Reauthorize Wait Timeout (Reauth Wait Timeout)

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

7.1.4.5.3 Authorization Grace Time (Auth Grace Timeout)

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

7.1.4.5.4 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.

7.1.4.6 Timers

7.1.4.6.1 Authorization Request

Used when awaiting a response to Authorization Requests.

7.1.4.6.2 Authorization Reject

Used after receipt of an Authorization Reject.

7.1.4.6.3 Authorization Grace

Used when determining when to reauthorize.

7.1.4.7 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.

[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.

[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]:

- clear Authorization Request timer;
- decrypt and record 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 a 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]:

- clear Authorization Request timer;
- decrypt and record 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.

7.1.5 TEK State Machine

The TEK state machine contains six states and nine events. The TEK state machine is presented below as both a state flow diagram and a state transition matrix. The CM MUST use the state transition matrix, together with the required actions in Section 7.1.5.6, as the definitive specification of protocol actions associated with each state transition.

Shaded states in Figure 7-3 ({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 upon 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 upon 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 a 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 7-3 illustrates the CM's scheduling of key refreshes in conjunction with its management of an SA's active TEKs.

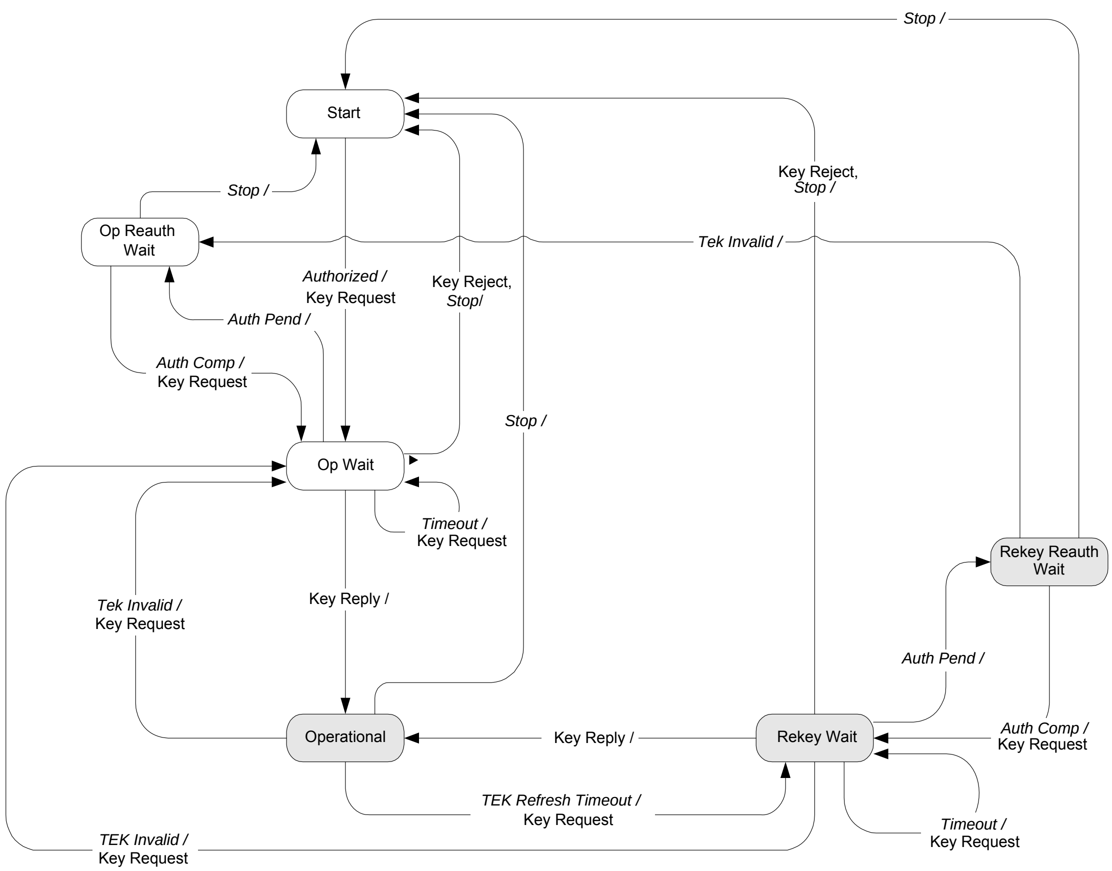

Figure 7-3 - TEK State Machine Flow Diagram

Table 7-2 - 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]					
{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]	

7.1.5.1 Brief Description of States

7.1.5.1.1 [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.

7.1.5.1.2 [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.

7.1.5.1.3 [Op Reauth Wait]

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

7.1.5.1.4 [Op]

The CM has valid keying material for the associated SAID.

7.1.5.1.5 [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.

7.1.5.1.6 [Rekey Reauth Wait]

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

7.1.5.2 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).

7.1.5.2.1 Key Request

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

7.1.5.2.2 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 Key Encryption Key (KEK) derived from the Authorization Key (see Section 11.4).

7.1.5.2.3 Key Reject

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

7.1.5.2.4 TEK Invalid

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

7.1.5.3 Brief Description of Events

7.1.5.3.1 {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 7.1.4.4.1.

7.1.5.3.2 {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 7.1.4.4.2.

7.1.5.3.3 {Auth Pend}

This event is sent by the Authorization FSM to a TEK FSM in order to place the TEK FSM in a wait state while the Authorization FSM completes reauthorization. See Section 7.1.4.4.3.

7.1.5.3.4 {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 7.1.4.4.4.

7.1.5.3.5 {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.,* a SAID's TEK key sequence number, contained within the received downstream packet's Baseline Privacy Extended Header 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.

7.1.5.3.6 {Timeout}

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

7.1.5.3.7 {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.

7.1.5.4 Brief Description of Timing Parameters

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

7.1.5.4.1 Operational Wait Timeout

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

7.1.5.4.2 Rekey Wait Timeout

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

7.1.5.4.3 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.

7.1.5.5 Timers

7.1.5.5.1 Key Request Retry

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

7.1.5.5.2 TEK Refresh

Used when to initiate a request for keys.

7.1.5.6 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

Baseline Privacy Key Management employs two MAC message types: BPKM-REQ and BPKM-RSP. The CMTS MUST support the BPKM formats as defined in this section. The CM MUST support the BPKM formats as defined in this section. [DOCSIS MULPI 3.0] defines the specific type values assigned to these messages (see also Table 7-3).

Table 7-3 - Baseline Privacy Key Management MAC Messages

Type Value	Message Name	Message Description
See [DOCSIS MULPI 3.0]	BPKM-REQ	Privacy Key Management Request [CM → CMTS]
See [DOCSIS MULPI 3.0]	BPKM-RSP	Privacy Key Management Response [CMTS → CM]

7.2.1 Packet Formats

One BPKM message is encapsulated in the Management Message Payload field of a single MAC management message.

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. When a packet is received with an invalid `Code` field, it SHOULD be silently discarded.

BPKM `Codes` (decimal) are assigned as follows:

Table 7-4 - Baseline Privacy Key Management Message Codes

Code	BPKM Message Type	MAC Management Message Name
0-3	Reserved	-
4	Auth Request	BPKM-REQ
5	Auth Reply	BPKM-RSP
6	Auth Reject	BPKM-RSP
7	Key Request	BPKM-REQ
8	Key Reply	BPKM-RSP
9	Key Reject	BPKM-RSP
10	Auth Invalid	BPKM-RSP
11	TEK Invalid	BPKM-RSP
12	Auth Info	BPKM-REQ
13	Map Request	BPKM-REQ
14	Map Reply	BPKM-RSP
15	Map Reject	BPKM-RSP
16-255	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 being 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 effect 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[14]).

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.

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

`Length`

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, the CMTS MUST ignore the additional octets. If the packet is shorter than indicated by the value of the `Length` field, the CMTS SHOULD silently discard it. If the packet contains more octets than indicated by the value of the `Length` field, the CM MUST ignore the additional octets. If the packet is shorter than indicated by the value of the `Length` field, the CM SHOULD silently discard it. The CMTS MUST set the value of the `Length` field to be in the range [0, 1490]. The CM MUST set the value of the `Length` field to be in the range [0, 1490].

`Attributes`

BPKM `Attribute`s 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 ...
+---------------+---------------+---------------+---------------+
```

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`

`Attribute List:`

Table 7-5 - Authorization Request Attributes

Attribute	Contents	See
CM-Identification	contains information used to identify CM to CMTS	7.2.2.5
CM-Certificate	contains the CM's Device certificate	7.2.2.18
Security-Capabilities	describes the CM's security capabilities	7.2.2.19
SAID	Initialization SAID or CM's Primary SAID	7.2.2.12

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 Manufacturer CA or CableLabs Mfg CA, and that signature can be verified by a CMTS that knows the corresponding public key.

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[15].

A SAID attribute contains the value of a 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.

7.2.1.2 Authorization Reply (Auth Reply)

Sent by the CMTS to a client CM in response to an Authorization Request, the Authorization Reply message contains an Authorization Key, the key's lifetime, the key's sequence number, and a list of SA-Descriptors, identifying the Primary and Static Security Associations that the CM is authorized to access, along with their particular properties (*i.e.,* type, cryptographic suite). The CMTS MUST encrypt the Authorization Key with the CM's public 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.

```
Code: 5
```

```
Attribute List:
```

Table 7-6 - Authorization Reply Attributes

Attribute	Contents	See
Auth-Key	Contains the Authorization Key, RSA encrypted with public key	7.2.2.7
Key-Lifetime	Authorization Key remaining lifetime	7.2.2.9
Key-Sequence-Number	Authorization Key sequence number	7.2.2.10
SA-Descriptor (one or more)	Compound attribute containing SAID and other properties.	7.2.2.23

7.2.1.3 Authorization Reject (Auth Reject)

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

```
Code: 6
```

```
Attribute List:
```

Table 7-7 - Auth Rej Attributes

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

[15] Formerly known as the Baseline Privacy Protocol.

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

7.2.1.4 Key Request

`Code: 7`

`Attribute List:`

Table 7-8 - Key Request Attributes

Attribute	Contents	See
CM-Identification	Contains information used to identify the CM to the CMTS	7.2.2.5
Key-Sequence-Number	Authorization key sequence number	7.2.2.10
SAID	Security Association ID	7.2.2.12
HMAC-Digest	Keyed message digest	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's authentication key is derived from the Authorization Key. See Section 11.4 for details.

7.2.1.5 Key Reply

`Code: 8`

`Attribute List:`

Table 7-9 - Key Reply Attributes

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

The TEK-Parameters attribute is a compound attribute containing all the keying material corresponding to a particular generation of a 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's authentication key is derived from the Authorization Key. See Section 11.4 for details.

7.2.1.6 Key Reject

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

`Code: 9`

`Attribute List:`

Table 7-10 - Key Reject Attributes

Attribute	Contents	See
Key-Sequence-Number	Authorization key sequence number	7.2.2.10
SAID	Security Association ID	7.2.2.12
Error-Code	Error code identifying reason for rejection of Key Request	7.2.2.15
Display-String (OPTIONAL)	Display string containing reason for Key Reject	7.2.2.6
HMAC-Digest	Keyed SHA message digest	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's authentication key is derived from the Authorization Key. See Section 11.4 for details.

7.2.1.7 Authorization Invalid

The Authorization Invalid message instructs the receiving CM to re-authorize with its CMTS.

`Code: 10`

`Attribute List:`

Table 7-11 - Authorization Invalid Attributes

Attribute	Contents	See
Error-Code	Error code identifying reason for Authorization Invalid	7.2.2.15
Display-String (OPTIONAL)	Textual description of failure condition	7.2.2.6

7.2.1.8 TEK Invalid

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.,* a SAID's TEK key sequence number, contained within the received packet's Baseline Privacy Extended Header element, is out of the CMTS's range of known, valid sequence numbers for that SAID.

`Code: 11`

`Attribute List:`

Table 7-12 - TEK Invalid Attributes

Attribute	Contents	See
Key-Sequence-Number	Authorization key sequence number	7.2.2.10
SAID	Security Association ID	7.2.2.12
Error-Code	Error code identifying reason for TEK Invalid message	7.2.2.15
Display-String (OPTIONAL)	Textual vendor-defined information	7.2.2.6
HMAC-Digest	Keyed SHA message digest	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's authentication key is derived from the Authorization Key. See Section 11.4 for details.

7.2.1.9 Authentication Information (Auth Info)

The Authentication Info message contains a single CA-Certificate `Attribute` field, which contains a Manufacturer CA or CableLabs Mfg CA certificate. The CM's Device certificate MUST have been issued by the certification authority identified by this certificate.

Authentication Information messages are usually informative: while the CM is required to transmit Auth Info messages as indicated by the Authentication state model (Section 7.1.4), the CMTS under some circumstances may ignore them.

```
Code: 12
```

```
Attribute:
```

Table 7-13 - Authentication Information Attributes

Attribute	Contents	See
CA-Certificate	Certificate of the Manufacturer CA or CableLabs Mfg CA	7.2.2.17

7.2.1.10 SA Map Request (MAP Request) [16]

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.

```
Code: 13
```

```
Attribute List:
```

Table 7-14 - SA Map Request Attributes

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

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

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

7.2.1.11 SA Map Reply (Map Reply)

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.

```
Code: 14
```

```
Attribute List:
```

Table 7-15 - 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	7.2.2.25
SA-Descriptor	Compound attribute containing the mapped SA's SAID and other properties	7.2.2.23

7.2.1.12 SA Map Reject (Map Reject)

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:

1. downstream traffic flow identified in the SA-Query Attribute is not being encrypted; or
2. the requesting CM is not authorized to receive that traffic.

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

```
Code: 15
```

```
Attribute List:
```

Table 7-16 - 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	7.2.2.25
Error-Code	Error code identifying reason for rejection of the SA Map Request	7.2.2.15
Display-String (OPTIONAL)	Textual reason for the SA Map Reject	7.2.2.6

7.2.2 BPKM Attributes

A summary of the format of the `Attribute` field is:

```
 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 7-17 - 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
19	Security-Capabilities
20	Cryptographic-Suite
21	Cryptographic-Suite-List
22	BPI-Version

Type	BPKM Attribute
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)
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)
29-50, 53-126	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 7-18.

Table 7-18 - Attribute Value Data Types

Type	Meaning
string	0 – 1487 octets
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 [ISO8859-1] character-set encoding. The CM MUST use only the following characters:

- Upper case letters, `A` to `Z` (0x41-0x5A)
- Lower case letters, `a` to `z` (0x61-0x7A)
- Digits, `0` to `9` (0x30-0x39)
- 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 three 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 specified is 106, 140, or 270 octets (length of DER-encoded RSAPublicKey, using F_4 as the public exponent and a 768-bit, 1024-bit, or 2048-bit public modulus respectively).

`string`

DER-encoded RSAPublicKey

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
- RSA-Public-Key

The 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 nor 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 contains either a ninety-six (96)- or a one-hundred twenty-eight (128)-octet value that is the Authorization Key encrypted with the CM's 768-bit or 1024-bit RSA public key. Details of the RSA encryption procedure are given in Section 11.5.

```
 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 or 128     | 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-180-2].

```
 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 Security Association ID (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`

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 a 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  = 33        | compound ...
+---------------+---------------+---------------+---------------+
```

`compound`

This field contains the following sub-attributes:

Table 7-19 - TEK-Parameters Sub-Attributes

Attribute	Contents	See
TEK	TEK, encrypted (two-key 3DES-EDE mode) with the KEK	7.2.2.8
Key-Lifetime	TEK Remaining Lifetime	7.2.2.9

Attribute	Contents	See
Key-Sequence-Number	TEK Sequence Number	7.2.2.10
CBC-IV	Cipher Block Chaining (CBC) Initialization Vector	7.2.2.14

7.2.2.14 CBC-IV

This attribute contains a Cipher Block Chaining (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 one-octet error code that provides further information about an Authorization Reject, Key Reject, Authorization Invalid, SA-MAP Reject [17] 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 7-20 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 the table (including zero) for the other message types. The CM MUST ignore error code values other than those defined in Table 7-20.

Table 7-20 - 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
4	Auth Invalid, TEK Invalid	Invalid Key Sequence Number

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

Error Code	Messages	Description
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

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

- 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;
- 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 per Section 13.3.2;
- The CM and the CMTS have incompatible security capabilities (see Section 7.2.1.1).

Entries in the CMTS MIB (see [DOCSIS OSSI 3.0]) 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 `Type`s defined within this specification and vendor-defined `Type`s, defined by the vendor identified in the preceding Manufacturer-ID sub-attribute.

7.2.2.17 CA-Certificate

This attribute contains a DER-encoded Manufacturer CA or CableLabs Mfg 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 Manufacturer CA or CableLabs Mfg 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 the 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 following sub-attributes:

Table 7-21 - 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 7-22 and a data authentication algorithm (encoded in the least significant octet) taken from Table 7-23. The CMTS MUST use a value that appears in Table 7-24 The CM MUST use a value that appears in Table 7-24 [18]

Table 7-22 - 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
4-255	Reserved

Table 7-23 - Data Authentication Algorithm Identifiers

Value	Description
0	No Data Authentication
1-255	Reserved

Table 7-24 - 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
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 ...
+---------------+---------------+---------------+---------------+
```

[18] This paragraph changed per SECv3.0-N-07.0409 #11 on 4.27.07 by KN.

`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 0). 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 7-25 - BPI-Version Attribute Values

Value	Description
0	Reserved
1	BPI+(security as defined in this specification)
2-255	Reserved

7.2.2.23 SA-Descriptor

This is a compound attribute whose sub-attributes describe the properties of a Security Association. 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 7-26.

Table 7-26 - SA-Descriptor Sub-Attributes[19]

Attribute	Contents	See
SAID	Security Association ID	7.2.2.12
SA-Type	Type of SA	7.2.2.24

[19] Changes to this table made per SECv3.0-N-07.0409 #9 on 4.27.07 by KN.

Attribute	Contents	See
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 7-27.

Table 7-27 - SA-Type Attribute Values

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

7.2.2.25 SA-Query [20]

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 7-28.

Table 7-28 - SA-Query Sub-Attributes[21]

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

[20] The SA mapping mechanism was used in earlier versions of DOCSIS to support dynamic SAs (see Annex C).
[21] Changes to this section made per SECv3.0-N-07.0409, #10 on 4/27/07 by KN.

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 7-29.

Table 7-29 - SA-Query-Type Attribute Values

Value	Description
0	Reserved
1	IP Multicast
2-127	Reserved
128-255	Vendor Specific

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:

- Zero or one instances of RSA-Public-Key (see Section 7.2.2.4);
- Zero, one or more instances of CA-Certificate (see Section 7.2.2.17);
- Zero or one instance of CVC-Root-CA-Certificate (see Section 7.2.2.29);
- Zero or one instance of CVC-CA-Certificate (see Section 7.2.2.30).

```
 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

This attribute contains a DER-encoded CVC Root 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 = 51  |             Length            | string ...
+---------------+---------------+---------------+---------------+
```

`Length`

Variable

`string`

DER-encoded CVC Root CA Certificate

7.2.2.30 CVC-CA-Certificate

This attribute contains a DER-encoded CVC 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 = 52  |             Length            | string ...
+---------------+---------------+---------------+---------------+
```

`Length`

Variable

`string`

DER-encoded CVC CA Certificate

8 EARLY AUTHENTICATION AND ENCRYPTION (EAE)

8.1 Introduction

The process of CM initialization is described in [DOCSIS MULPI 3.0]. This section specifies requirements on the CM and the CMTS for securing the CM initialization process. This specification introduces 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

The CMTS uses TLV type 6 in the MDD MAC message to signal EAE to CMs. The CM determines whether it is required to perform EAE by detecting the MDD MAC message and inspecting TLV type 6.

When the CMTS is configured to enable EAE (see Section 8.4), the CMTS MUST include TLV type 6 in the MDD with its value set to 1 to signal to CMs that they must perform EAE. When the CMTS is configured to disable EAE, the CMTS MUST include TLV type 6 in the MDD with its value set to 0, to signal to CMs that they do not perform EAE.

If the CM detects a valid MDD message before or during initial ranging, and the MDD contains TLV type 6 with a nonzero value, then the CM MUST start EAE by sending a {Initiate Authentication} event to the BPI+ Auth State Machine (see Section 7) immediately following Ranging completion. If the CM detects no MDD message before or during initial ranging, or the detected MDD message contains no TLV type 6, or the detected MDD contains TLV type 6 with value set to zero (0), then the CM MUST NOT start EAE. The CM follows procedures specified in [DOCSIS MULPI 3.0] 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.

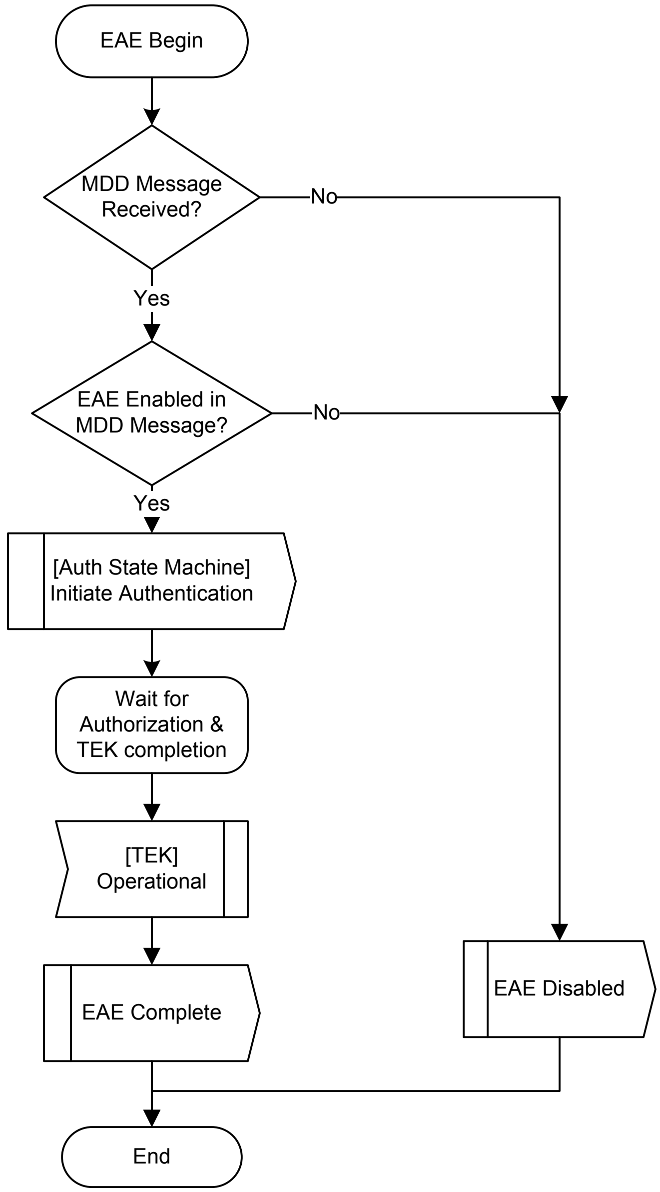

Figure 8-1 - EAE Signaling Flow Chart for CM

8.3 EAE Encryption

A CMTS that has completed EAE Authorization with a CM MUST transmit all subsequent unicast DOCSIS data PDUs that are not otherwise assigned to a Security Association (SA) on the CM's primary SA, until such time BPI+ is disabled. [22]

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:

[22] Revised this paragraph per ECN SECv3.0-N-06.0287-2 by GO on 10/6/06.

- 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).

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 MULPI 3.0]).

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 1.0/1.1/2.0 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, then:

- following successful completion of the ranging process, the CMTS MUST drop all PDUs (*i.e.,* frames with FC type 00) [DOCSIS MULPI 3.0] 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).

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 [DOCSIS MULPI 3.0]. 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).

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 (Section 8.4.4).

8.4.3 DHCP Relay and EAE Enforcement Verification

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 version 3.0 or later [DOCSIS MULPI 3.0]; and
- the CM has not successfully completed EAE; and
- the CM is not on the EAE Exclusion List (see 8.4.5).

8.4.4 CMTS and CM Behavior when EAE is Disabled

When EAE is disabled, then:

- the CMTS MUST allow a CM to proceed with the Initialization process [DOCSIS MULPI 3.0] 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 MULPI 3.0];
- 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, then:

- the CMTS MUST allow the CM to proceed with the Initialization process [DOCSIS MULPI 3.0] 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 proceed to the next step in its initialization

process as described in [DOCSIS MULPI 3.0]. 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 RFI 2.0].

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 MULPI 3.0]. 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 [DOCSIS MULPI 3.0] 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 A–1, 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 Re-Authorize 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 Authorization lifetime, then the CM MUST immediately set the Authorization Grace Time to 1 second less than half the Authorization lifetime. If the configuration file does not contain BPI+ settings, or BPI+ settings are present and have default values listed in Table A–1, then the CM’s state machines MUST continue to operate as normal EAE (see Annex A.2). [23]

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. [24]

[23] Changes made to paragraph per SECv3.0-N-07.0387-1 on 4/26/07 by KN.
[24] Changes made to REQ per SECv3.0-N-07.0379.1, #2 on 4/26/07 by KN.

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 CM MUST send a {Initiate Authentication} event to the Authorization State Machine (see Section 7.1.4) 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 MULPI 3.0], and initialization of Baseline Privacy operations completion is defined in Section 5.3.1.
- 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 MULPI 3.0] 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 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 [DOCSIS MULPI 3.0]. As the lightweight DHCPv6 authentication protocol relies on DHCPv6 messages to distribute Reconfigure keys to the CMs, it is essential that DHCPv6 messages be encrypted. If a CM is provisioned to accept DHCPv6 Reconfigure messages, then EAE should be enabled, in order to protect 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, 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, 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, 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 [RFC 2347], TFTP blocksize option [RFC 2348] and TFTP timeout interval option [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 MULPI 3.0]). The CMTS MUST also log an event.

9.4.2.5 TFTP Options for CM's MAC 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 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 (Annex B). 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. [25]

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 are communicated to the CMTS in REG-REQ-MP messages (see [DOCSIS MULPI 3.0]). To maintain confidentially of these parameters, when EAE is enabled the CM MUST encrypt the REG-REQ-MP message (see Section 6.5). 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 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

[25] Paragraph changed per SECv3.0-N-07.0409-1, #2 on 4/26/07 by KN.

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. Source IP addresses are considered assigned by the operator when they are provisioned via DHCP messaging or identified by parameters in the configuration file.[26]

The CM configuration file contains TLV encodings [DOCSIS MULPI 3.0] that can be 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, or
3. A Subscriber Management CPE IPv4 or IPv6 Encoding.

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:[27]

1. matches the Subscriber Management CPE IPv4 or IPv6 Encoding [DOCSIS MULPI 3.0] 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 [DOCSIS MULPI 3.0] 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 DHCP-ACK messages sent in response to an upstream DHCP-REQ received from a CM's SID.
    b. use of a CMTS initiated DHCP LEASE-QUERY response that provides the assigned IP address for a host source MAC address.
    c. the CMTS extracts assigned IP address information from an IPv6 prefix delegation option.

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 Subscriber Management CPE IPv4 or IPv6 Encoding, SAV Group ID Encoding and Static SAV Prefix Encoding for identification of operator assigned IP addresses.

When Source Address Verification is enabled the 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. [28]

The CMTS provides per-CM statistics of the number of packets discarded due to SAV failure.

[26] Paragraph changed per SECv3.0-N-07.0409-1, #1 on 4/26/07 by KN.
[27] Revised this sentence and subsequent steps below, per ECN SECv3.0-N-06.0292-2 by GO on 10/13/06.
[28] Revised this paragraph per ECN SECv3.0-N-06.0292-2 by GO on 10/13/06.

9.7 Address Resolution Security Considerations

Address Resolution Protocol (ARP) [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 [RFC 2461] 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, 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, or by acting in proxy for the address resolution target.

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 MULPI 3.0] 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 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 the 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 10-1.

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 active. If the newer key has not been implicitly acknowledged, the CMTS MUST use the older of the two active AKs.

Figure 10-1 illustrates the CMTS's use of AKs.

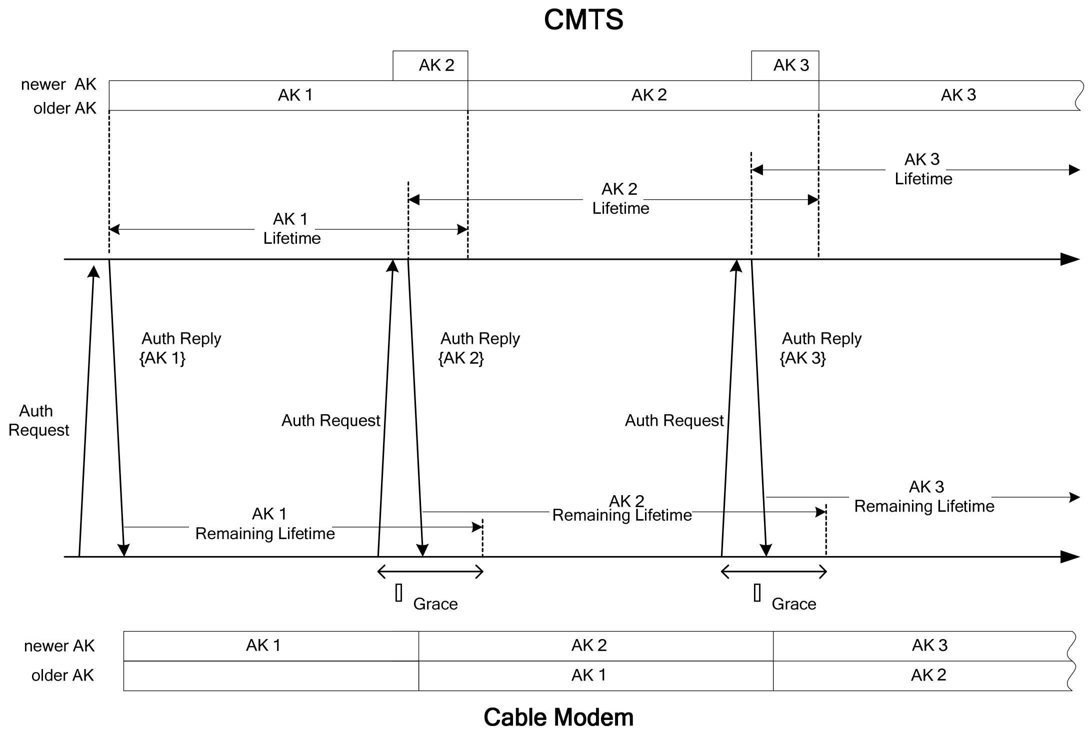

Figure 10-1 - Authorization Key Management in CMTS and CM

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 10-2 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.

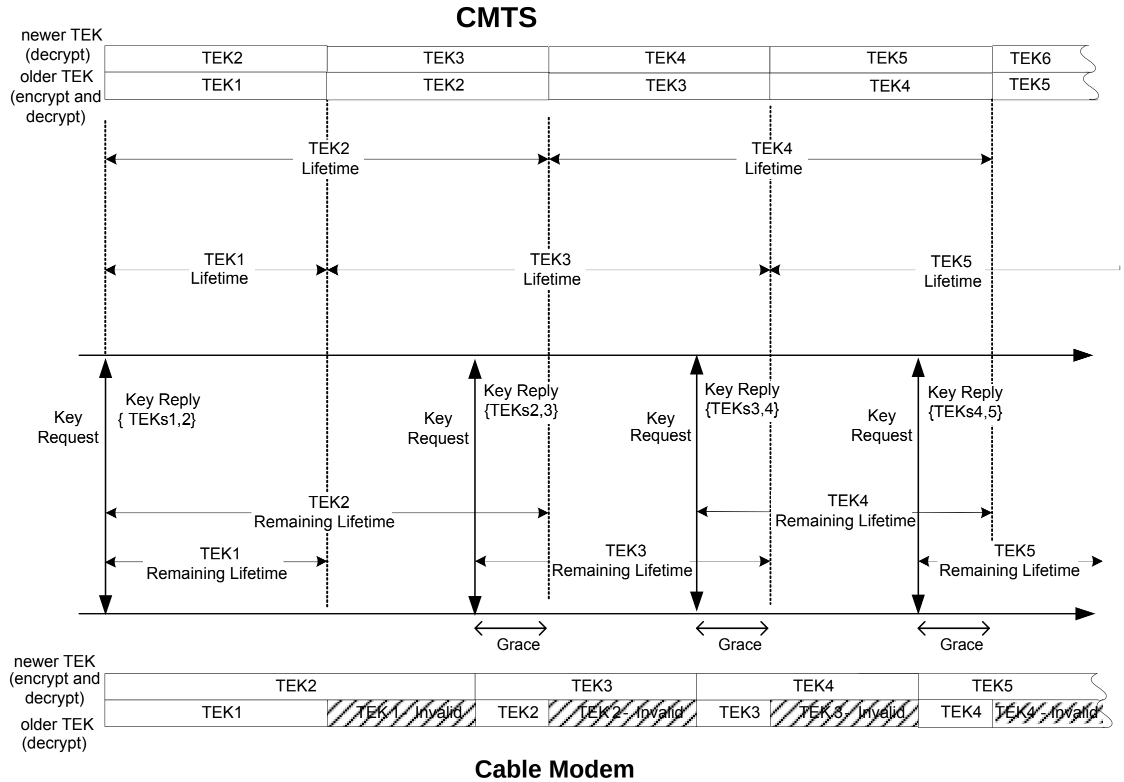

Figure 10-2 - TEK Management in CMTS and CM

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 7.1.4) 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 10-1 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 a 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. Traffic already queued for transmission MUST use an unexpired TEK;
- MUST be able to decrypt downstream traffic encrypted with either active TEK.

10.3 Authentication of Dynamic Service Requests

10.3.1 CM

If EAE or 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 EAE or 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 [NIST-800-38A] of either the Data Encryption Standard (DES) algorithm [FIPS-46-3], or the Advanced Encryption Standard (AES) algorithm [FIPS-197] to encrypt the Packet Data field, RF MAC PDU Frames. The CM MUST use the CBC mode [NIST-800-38A] of either the Data Encryption Standard (DES) algorithm [FIPS-46-3], or the Advanced Encryption Standard (AES) algorithm [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 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 MUST support fifty-six (56)-bit DES. The CMTS MAY support forty (40)-bit DES.

BPI+ supports forty (40)-bit DES principally to permit interoperability with forty (40)-bit DOCSIS 1.0 hardware upgraded to run BPI+. 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, it 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, it 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 1.0 or 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 = Ek1[Dk2[Ek1[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 [RFC 2104] with the SHA-1 hash algorithm [FIPS-180-2]. When creating or verifying the HMAC-Digest Attribute, the CM MUST use the HMAC message authentication method [RFC 2104] with the SHA-1 hash algorithm [FIPS-180-2].

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. The CMTS should follow practices recommended in [RFC 1750] 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 Public-Key Encryption of Authorization Key

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 [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. 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.

Note: Baseline Privacy [SCTE 22-2] employed the encryption scheme described in version 1.5 of the PKCS #1 standard [RSA1]. This is the same scheme as RSAES-PKCS1-v1_5 in [RSA3]. In order to maintain backwards compatibility, CMTSs MUST revert to RSAES-PKCS1-v1_5 for encrypting the authorization key when falling back to BPI.

In order to interoperate with earlier versions of this specification, the CMTS MUST support 768-bit and 1024-bit key moduli.

11.6 Digital Signatures

All DOCSIS certificates use the RSA signature algorithm [RSA3] with the SHA-1 hash [FIPS-180-2].

Manufacturer CAs use signature keys with a modulus in the range 1024 bits ≤ modulus ≤ 2048 bits.

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 MULPI 3.0]).

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, 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 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:

$$\kappa = \kappa_1, \kappa_2, \ldots, \kappa_\sigma.$$

Likewise, the input message is denoted by M and the ith word of the input message by M_i:

$$M = M_1, M_2, \ldots, M_\sigma.$$

To describe $\mathcal{H}^1$, the following definitions are needed:
For any even positive integer n, S_n is the set of n integers:

$$Sn = \{-n/2, \ldots, 0, \ldots, (n/2)-1\}$$

For example,

$$S_{2^{16}} = \{-215, \ldots, 0, \ldots, 215-1\}$$

is the set of signed 16 bit integers.

For any integer z, z smod 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 smod 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 $S^{\sigma}_{2^{16}} \times S^{\sigma}_{2^{16}}$ 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 $\kappa, M \in S^{\sigma}_{2^{16}}$,

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,

$$\mathcal{H}^1(\kappa, M) = \left(\left(\left(\sum_{i=1}^{\sigma} \kappa_i \cdot M_i\right) \sigma \bmod 2^{32}\right) \bmod p\right) \bmod 2^{16}$$

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 smod 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 mod 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 $a \in \{0, \ldots, 2^{15}-1\}$ and $b \in \{0, \ldots, 2^{16}-1\}$.

From the modular equation:

$$2^{16}a + b \equiv (2^{16}a + b - a(2^{16}+1)) \bmod (2^{16} + 1)$$

it follows that x ≡ b – a, mod p. The quantity b – a is in the range {-215+1, …, 216 – 1}. Therefore, if (b – a) ≥ 0 then x mod p = b – a. If b – a < 0 then x mod p = b – a + p.

Case 2: If x < 0, then x = 216a + b – 232, where a ∈{215, …, 216 – 1} and b ∈{0, …, 216 – 1}.

From the modular equation:

216a + b – 232 ≡ (b + 216a – a(216+1) – 232 + 216(216 + 1)) mod (216 + 1)

it follows that x ≡ (b – a + 216) mod p. The range of the quantity b – a + 216 is given by:

1 ≤ b – a + 216 ≤ 217 – 215 - 1 ≤ 2p – 1

Therefore, if b – a + 216 < p, then x mod p = b – a+ 216. If b – a + 216 ≥ p,
then, x mod p = b – a + 216 – p.

Step 3. This step takes an element of Zp and reduces it modulo 216. 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, σ, n]. $\mathcal{H}^n$ takes a key of $\sigma + n - 1$ words. Let $\kappa = \kappa_1, \ldots, \kappa_{\sigma+1}$. Furthermore, define $\kappa^{(q)}$ to be the σ words of κ, starting with κ_q, *i.e.*, $\kappa^{(q)} = \kappa_q, \ldots, \kappa_{\sigma+q-1}$. 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 $\mathcal{H}^1(\kappa^{(1)}, M), \ldots, \mathcal{H}^1(\kappa^{(n)}, M)$ and concatenating the results. That is:

$$\mathcal{H}^n(\kappa, M) = \mathcal{H}^1(\kappa^{(1)}, M) \circ \mathcal{H}^1(\kappa^{(2)}, M) \circ \ldots \circ \mathcal{H}^1(\kappa^{(n)}, M).$$

11.7.2 MMH[16, σ, 4]

It follows directly from Section 11.7.1.2 that:

$$\mathcal{H}^4(\kappa, M) \equiv \text{MMH}[16, \sigma, 4](\kappa, M)$$

$$\equiv \mathcal{H}^1(\kappa^{(1)}, M) \circ \mathcal{H}^1(\kappa^{(2)}, M) \circ \mathcal{H}^1(\kappa^{(3)}, M) \circ \mathcal{H}^1(\kappa^{(4)}, M)$$

11.7.2.1 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.
- It a key is calculated that is larger than needed for a particular message, truncate the key until it is the correct length.

11.7.3 Definition of MMH-MAC

The MMH-MAC is defined in a manner similar to PacketCable [PKT-SEC][29]; 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*:,

$$\text{MMH-MAC}(\kappa, M, \Pi) = \mathcal{H}(\kappa, M) + \Pi$$

where the addition is mod 2^n.

11.7.4 Calculating the MMH-MAC

The following is a generic algorithm used for creating an MMH-MAC that can be used to protect a message. 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, S1 and S2.
2. The Sender and Receiver share a seed, Δ1, which may be public.

[29] Note, however, that the definition is only identical to the PacketCable definition in the case that the output is one 16-bit word.

3. The Sender and Receiver optionally share a second seed, Δ2, which may be public. In the absence of the second seed, they treat Δ2 as identical to Δ1.
4. The message to be protected is M.
5. F(<secret>, <seed>) is a pseudo-random generator whose output depends on a shared secret (<secret>) and a seed (<seed>) that may be public.
6. The length of the MMH MAC is L octets.

Steps to create MMH-MAC:

1. Calculate a keystream, κ = F(S1, Δ1), of sufficient length to generate the MMH function H over (M, κ). That is, the length of κ is the length of M plus the output length of the MMH function.
2. Calculate the output of the MMH function H = MMH(κ, M).[30]
3. Accumulate a value A of length L by XORing every L octets from step 2, padding with zeroes as necessary (*i.e.,* add zeroes to the end of H to pad it out to a length that is an integral multiple of L).
4. Concatenate A + S2.
5. Calculate the first L octets of F(A + S2, Δ2).
6. Use the output of step 5 as the one time pad Π in the calculation of the MMH-MAC as defined in Section 11.7.3.

11.7.5 MMH Key Derivation for CMTS Extended MIC[31]

The CMTS and the backend provisioning system share a secret key, $\kappa_{CMTS\text{-}EMIC}$. This is equivalent to the shared secret *S* in Section 11.7.4.

From $\kappa_{CMTS\text{-}EMIC}$, the CMTS MUST derive two shared secrets, equivalent to S_1 and S_2 in Section 11.7.4:

$S_1 = S[0]\ )S[2]\ )\ S[4]\ )\ \ldots\ )\ S[n\text{-}2]$

$S_2 = S[1]\ )\ S[3]\ )\ S[7]\ )\ \ldots\ )\ S[n\text{-}1]$

where S[i] is the ith octet of S (with respect to 0) and S contains n octets.

The CMTS MUST use the ASCII-encoded string "CMTS-EMIC" as the value of Δ_1 in the calculation of the MMH-MAC.

The CMTS MUST use the ASCII-encoded string "CMTS-EMIC-PAD" as the value of Δ_2 in the calculation of the MMH-MAC.

The CMTS MUST use the function F as defined in Section 11.8 as the pseudo-random number generator when calculating the MMH-MAC. [32]

11.7.6 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, the authors recommend that the shared secret:

[30] Changes to steps made per SECv3.0-N-07.0409.1 #4 on 4/26/07 by KN.
[31] Revised this section per ECN SECv3.0-N-06.0291-1 by GO on 10/4/06.
[32] Paragraph that follows this deleted per SECv3.0-N-07.0409-1, #3 on 4/26/07 by KN.

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).
2. be of length 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.

11.8 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 IN THE CM

CMs MUST store and maintain the CM certificate RSA private/public key pair. The CM MUST store the CM certificate private key in a manner that deters unauthorized disclosure and modification. Also, CMs SHOULD prevent debugger tools from reading the CM 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 shall be of production-grade quality, which shall include 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 shall be implemented as a production grade multiple-chip embodiment (*i.e.,* an IC printed circuit board, a ceramic substrate, etc.);
- The module shall 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 CMCI]) 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.

13 BPI+ X.509 CERTIFICATE PROFILE AND MANAGEMENT

DOCSIS employs [X.509] version 3 digital certificates for authenticating key exchanges between CM and CMTS [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 3280].

The DOCSIS certificate profile draws extensively from the Secure Electronic Transaction (SET) system [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, depicted in Figure 13-1 and Figure 13-2, consists of a three-level hierarchy of trust supporting three types of certificates:

- DOCSIS Root CA certificate;
- Manufacturer CA (distributed model) or CableLabs Mfg CA (centralized model) certificates;
- CM Device certificates.

The DOCSIS Root Certificate Authority serves as the root CA. The root CA issues the CableLabs Mfg CA certificate (which is maintained by CableLabs) and Manufacturer CA certificates (which are maintained by manufacturers).

DOCSIS supports a "Centralized Model" (Figure 13-1), in which the CableLabs Mfg CA issues certificates to CMs and a "Distributed Model" (Figure 13-2), in which Manufacturer CAs issue certificates to CMs.

A single manufacturer may maintain multiple Manufacturer CAs (*e.g.,* a different CA for each manufacturing plant). A single manufacturer may use both the "Centralized Model" and the "Distributed Model" at the same time.

The DOCSIS Root CA certificate is installed on the CMTS in order to support CM authentication. In the future, it is anticipated that the CableLabs Manufacturer Root CA certificate may be used in addition to, or instead of, the DOCSIS Root CA to issue and validate Manufacturer CA certificates.

The DOCSIS Root CA also serves as the root CA that issues Code Verification Certificates (CVC) for the Secure Software Download process specified in Section 14. In the future, it is anticipated that the CableLabs CVC Root CA may be used in addition to, or instead of, the DOCSIS Root CA to validate CVCs issued by an intermediate CA.

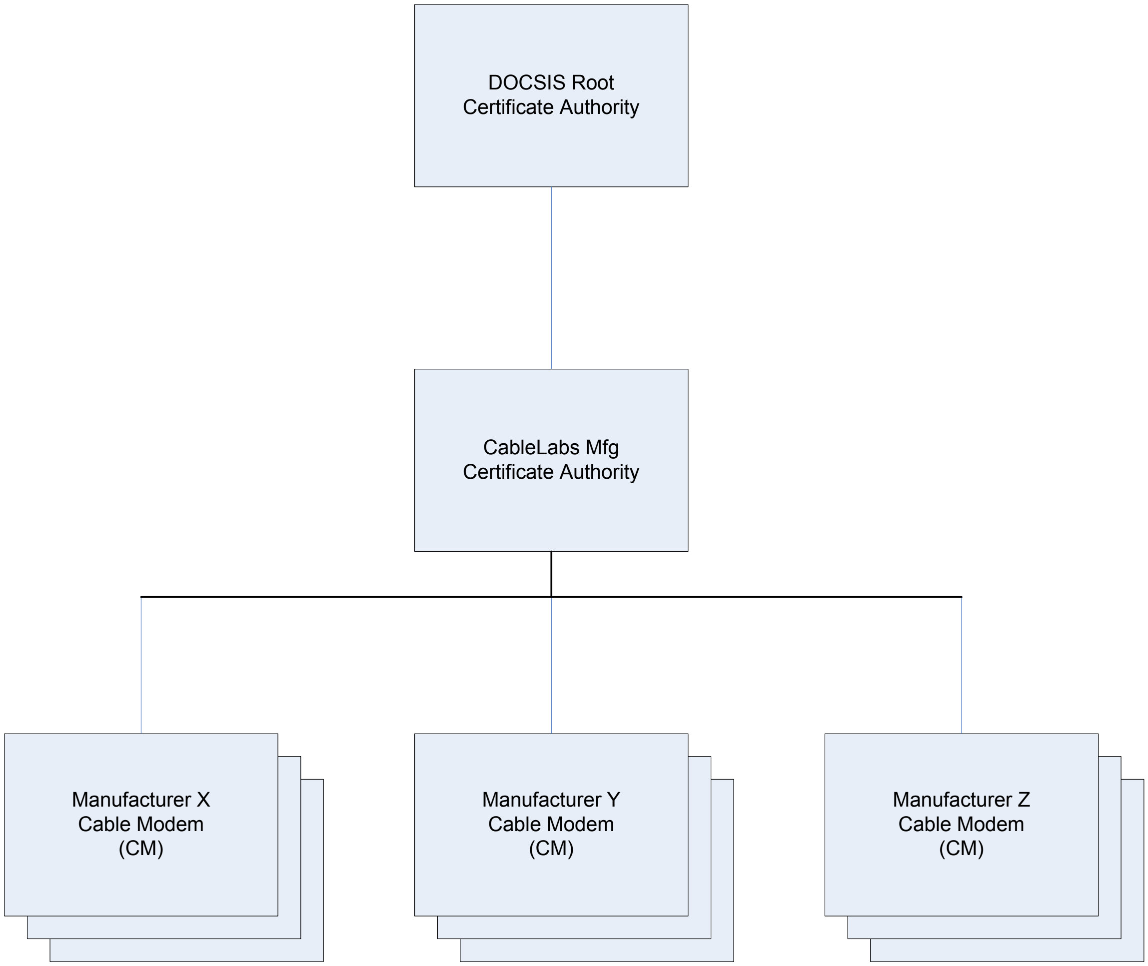

Figure 13-1 - The Centralized Model of the DOCSIS Certificate Management Architecture

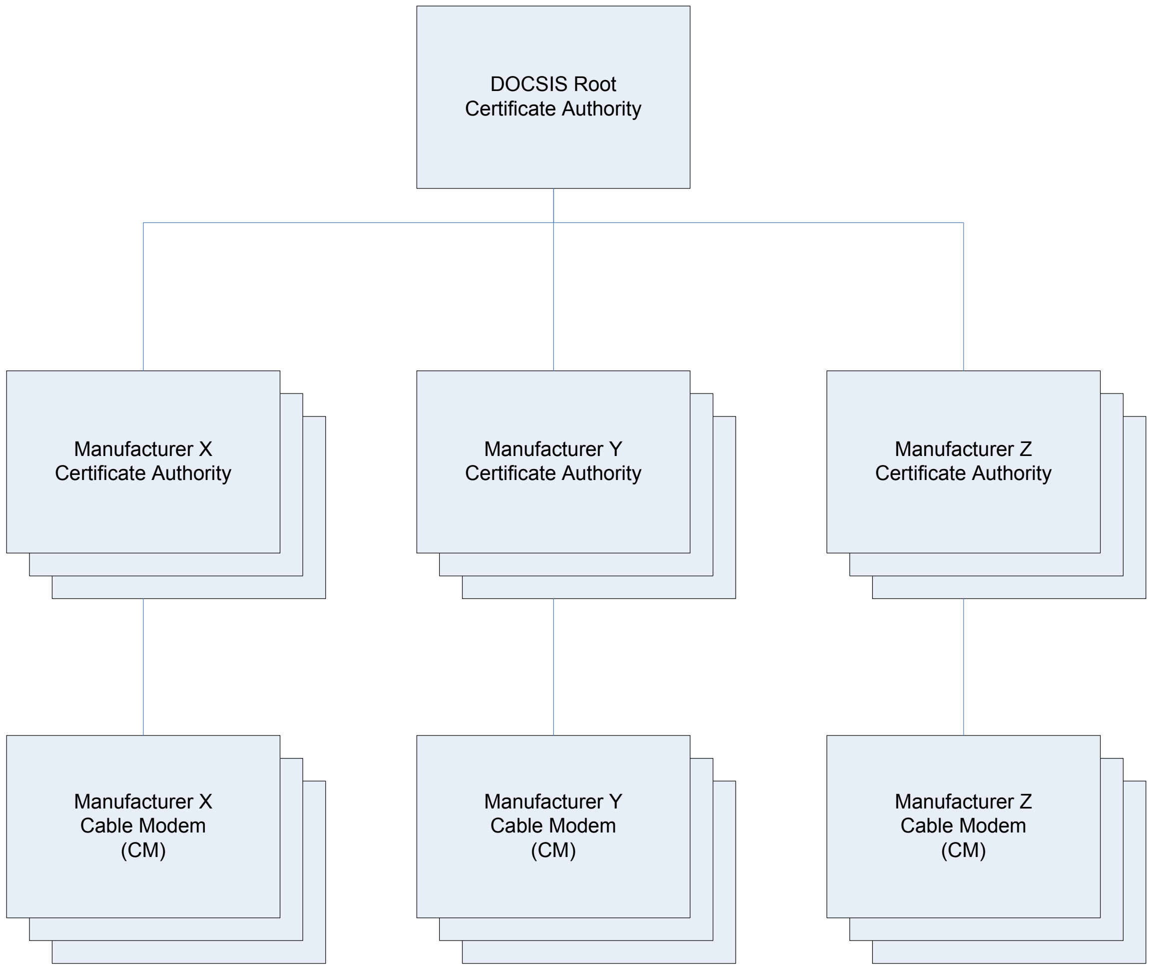

Figure 13-2 - The Distributed Model of the DOCSIS Certificate Management Architecture

The DOCSIS Root CA is held under tight access controls. The organization responsible for DOCSIS certification is responsible for maintaining the DOCSIS Root CA. The DOCSIS 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.

The organization maintaining the DOCSIS Root CA shall define a protocol to issue the Manufacturer CA certificates, however, that protocol is outside the scope of this specification.

Manufacturers that utilize the Distributed Model of the DOCSIS Certificate Hierarchy are responsible for maintaining their own Manufacturer CA(s), from which they issue CM certificates. A single manufacturer may maintain multiple Manufacturer CAs. Manufacturers that use the Centralized Model of the DOCSIS Certificate Hierarchy obtain CM certificates from the CableLabs Mfg CA. Protocols for requesting certificates from a Manufacturer CA and distributing the resulting certificates to the receiving Cable Modems are internal to that manufacturer and are outside the scope of this specification. A Manufacturer CA may generate and distribute CRLs to operators. The manner in which these CRLs are distributed is outside the scope of this specification.

13.2 Cable Modem Certificate Storage and Management in the CM

CMs MUST have factory-installed CM certificates. The RSA public key in the CM Certificate MUST be the same as the RSA public key in the BPKM Attributes.

The DOCSIS Root CA public key for the CVC verification MUST be placed into the CM's non-volatile memory. Currently, the DOCSIS Root CA issues the CVCs. However, in the future, the CableLabs CVC Root CA may be used to issue and validate CVCs and therefore the CM MUST be capable of updating or replacing the DOCSIS Root CA public key via the DOCSIS code download file (see Section 14).

The CA certificate (Manufacturer CA or CableLabs Mfg CA) that signed the CM certificate MUST be stored in the CM's non-volatile memory. The CM MAY be capable of updating or replacing the Manufacturer CA certificate via the DOCSIS code download file (see Section 14). The CA certificate MAY be embedded into the CM software.

The CM MUST be able to process certificate serialNumber 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 certificate's certification path. This path will typically consist of three chained certificates: the CM certificate, the Manufacturer CA certificate or CableLabs Mfg CA certificate, and the DOCSIS Root CA certificate (see Section 13.1). Validating the chain follows the "Basic Path Validation" rules defined in [RFC 3280].

[RFC 4131] requires that CMTSs support administrative controls that allow the operator to override certification chain validation by identifying a particular CA or CM 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 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.

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. Any certificate transmitted by a CM in an Auth Info or Auth Request message has 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.

13.3.1 CMTS Certificate Management Model

The CMTS holds copies of the Root CA, Manufacturer CA or CableLabs Mfg CA, and CM certificates, 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 Manufacturer CA and CableLabs Mfg 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 Manufacturer CA or CableLabs Mfg 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 Manufacturer 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 Certificates in the Authorization Requests it receives from CMs. CM Certificates are issued by a Manufacturer CA or the CableLabs Mfg CA. Thus, the CMTS MUST mark CM 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 certificates using Basic Path Validation rules defined in [RFC 3280] 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, Trusted, or Valid certificate that has not been revoked as defined by the Basic Path Validation section in [RFC 3280]; and
2. the current time falls within the validity period of each Chained or Root certificate within the certificate chain (BPI+ does not require the nesting of validity periods, *i.e.,* a certificate's entire validity period need not fall within the validity period of it's issuing certificate – however, see paragraph below this list); and
3. the certificate is not identified as revoked (see Section 13.4); and
4. in the case of a CM 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 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 Manufacturer CA Certificate, if the KeyUsage extension is present, the keyCertSign bit is turned on.

Whether criterion 2 above is ignored MUST be subject to CMTS administrative control.

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 3280]) 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).

13.4.1 Certificate Revocation Lists

[RFC 3280] defines a method for revoking certificates using [X.509] Certificate Revocation Lists (CRLs).

Figure 13-3 shows a framework for managing and distributing CRLs. A CRL is a digitally signed, time stamped 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 CRL Repository is outside the scope of this specification.

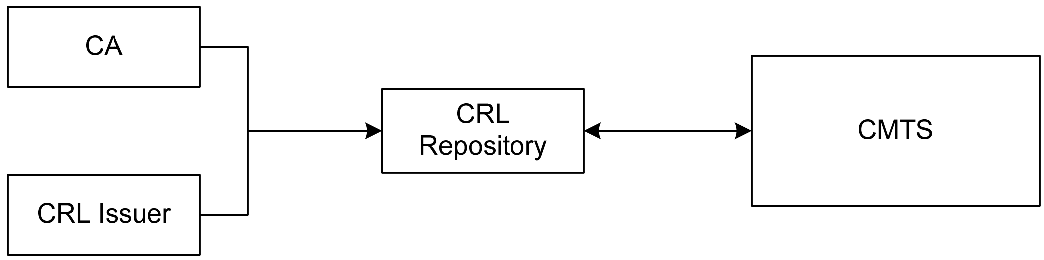

Figure 13-3 - CRL Framework

The CMTS retrieves CRL entries from one or more CRL Repositories and uses this information to verify if a certificate received during the CM registration process is revoked.

13.4.1.1 Certificate Revocation List Format

Table 13-1 below summarizes the CRL fields of a [X.509] version 2 CRL.

Table 13-1 - X.509 CRL Fields

X.509v2 CRL field	Description
tbsCertList.version	Indicates [X.509] CRL version. Always set to version 2 (integer value 1)
tbsCertList.signature	Signature algorithm used to sign the CRL
tbsCertList.issuer	Entity that signs and issues the CRL
tbsCertList.thisUpdate	Issue date of the CRL
tbsCertList.nextUpdate	Date by which the next CRL will be issued
tbsCertList.revokedCertificates.userCertificate	Individual certificate serial number
tbsCertList.revokedCertificates.revocationDate	Date the certificate is revoked
tbsCertList.revokedCertificates.crlEntryExtensions	Additional attributes associated with individual CRL entries
tbsCertList.crlExtensions	Extensions associated with the CRL
signatureAlgorithm	Signature algorithm used to sign the CRL
signatureValue	Digital Signature computed on the tbsCertList

All of the above fields conform to [RFC 3280].

13.4.1.2 CMTS CRL Support

The CMTS MUST support retrieval of CRL files formatted as defined in [RFC 3280]. 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 certificate is revoked during the certificate validation process specified in Section 13.3.2.

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, it MUST log an event [DOCSIS OSSI 3.0] and continue to use its current CRL. If the CRL does not contain the nextUpdate value the CMTS MUST refresh the CRL according to the configured value as defined in [DOCSIS OSSI 3.0]. The CMTS MUST make CRL data persistent.

13.4.2 Online Certificate Status Protocol

[RFC 2560] 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 certificate (see Figure 13-4). 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.

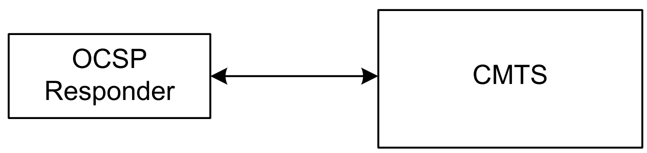

Figure 13-4 - OCSP Framework

The CMTS MUST be capable of acting as an OCSP client as defined in [RFC 2560]. 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 it MUST NOT retrieve the revocation status from the cache once the nextUpdate time for that certificate has passed. 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, it MUST send an OCSP request when a CA certificate or CM certificate is obtained using the Authentication Information message, or Authentication Request message respectively, unless there is a valid certificate status in the cache. If the CMTS is unable to retrieve the OCSP status for a certificate or if the retrieved status is “unknown” the CMTS MUST log an event [DOCSIS OSSI 3.0] and treat the certificate status as “good " When the CMTS is attempting to communicate with the OCSP Responder, the exchange should not significantly delay the CM provisioning process. If no response is received within one second, the CMTS MUST proceed as if a response with the status "good" has been received. [33]

The CMTS MUST support OCSP over HTTP as described in appendix A of [RFC 2560]. 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 OSSI 3.0].

[33] Revised this paragraph per ECN SECv3.0-N-06.0322-1 by KN on 11/17/06.

14 SECURE SOFTWARE DOWNLOAD

14.1 Introduction

DOCSIS 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.

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 should have a means to authenticate that the originator of any download code is a known and trusted source;
- The CM should 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 should 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 Public Key in the CM (optional);
- Support updating the Manufacturer CA Certificate(s) in the CM (optional);
- Support updating the CableLabs CVC Root CA Certificate in the CM (optional). This certificate is currently not being used. It might be used in the future use if the manufacturer CVC is signed by the CableLabs CVC Root CA Certificate chain;
- Support updating the CableLabs CVC CA Certificate in the CM (optional). This certificate is currently not being used. It might be used in the future use if the manufacturer CVC is signed by the CableLabs CVC Root CA Certificate chain.

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 DOCSIS Root CA. The manufacturer's 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's signature. If a second signature is present, the CM verifies both signatures with a certificate chain that extends up to the DOCSIS 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. 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 14-1 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.

In DOCSIS, each CM receives a public key from the DOCSIS Root Certificate Authority. The code manufacturer builds the code file by signing the code image using a DOCSIS PKCS#7 digital signature structure with a DOCSIS 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 TFTP server as-is, or of adding its signature and operator CVC to the code file. During the code upgrade process, the CM will retrieve the code file from the TFTP server and verify the new code image before installing it.

Currently, the DOCSIS Root CA issues both Manufacturer CA certificates and CVCs. In the future the CableLabs Manufacturer Root CA certificate may be used to issues and validate Manufacturer CA certificates and the CableLabs CVC Root CA may be used to issue and validate CVCs.

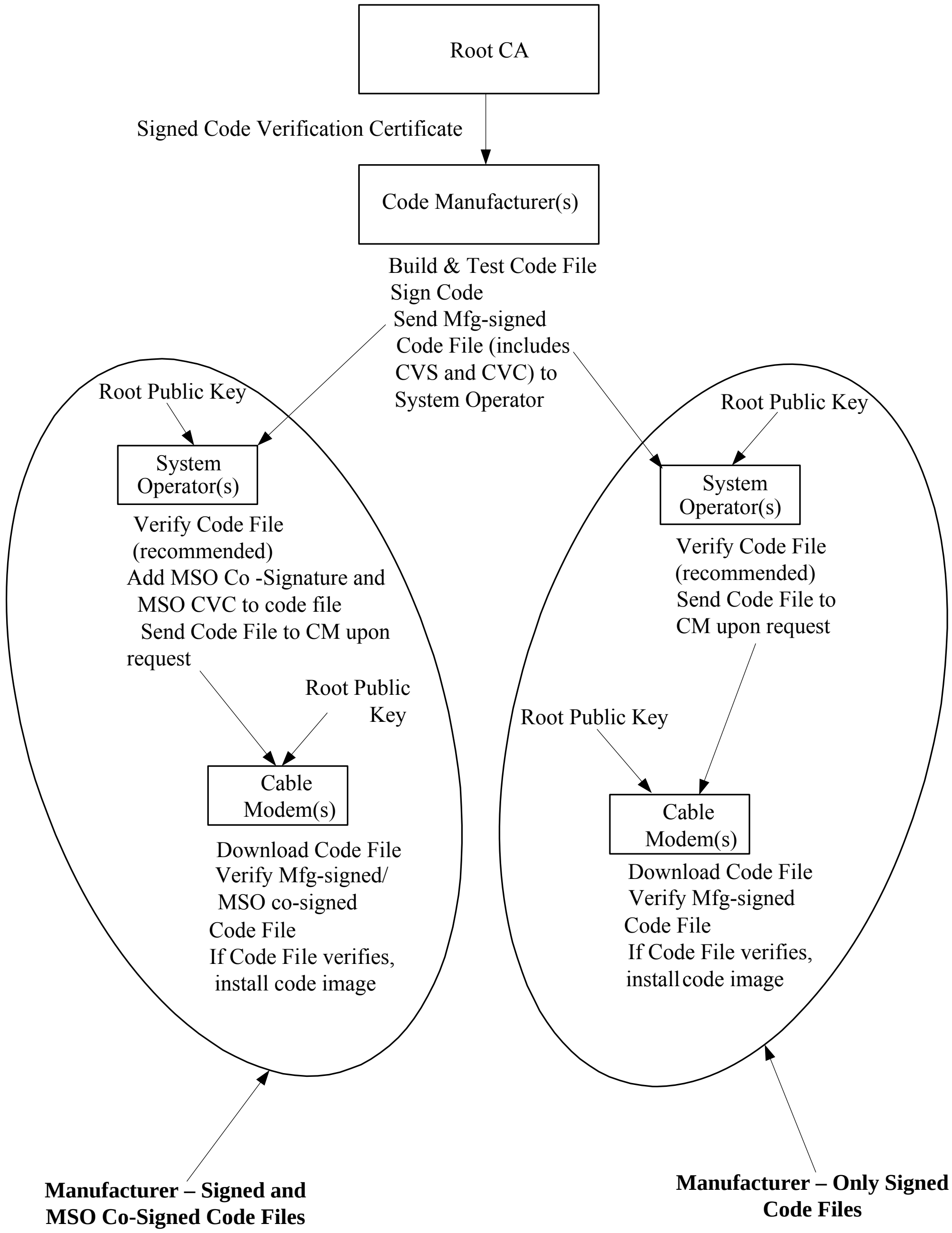

Figure 14-1 - Typical Code Validation Hierarchy

14.3 Software Code Upgrade Requirements

The following sections define requirements of the CM software code upgrade verification process. All DOCSIS 1.1, 2.0 or 3.0 code upgrades are prepared and verified as described. All DOCSIS 1.1, 2.0 or 3.0 CMs MUST verify code upgrades according to this specification regardless of whether Baseline Privacy is enabled or disabled.

14.3.1 Code File Processing Requirements

The code file format is defined in Appendix III.

The CM MUST reject the DOCSIS [PKCS#7] 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 1024, 1536 and 2048 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 DOCSIS Root CA Public key with the DOCSIS Root CA Public key in the SignedContent field, if one was present.

If the code download and installation is successful, then the CM MUST replace its currently stored Manufacturer CA certificate(s) with the Manufacturer CA certificate(s) received in the SignedContent field, if any were present.

If the code download and installation is successful, then the CM MUST store the CVC Root CA certificate and the CVC CA certificate for future use if they are 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 and 14.3.2.2).

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: The CM MUST verify that the manufacturer name in the manufacturer's CVC subject field exactly matches the manufacturer name stored in the CM's non-volatile memory by the manufacturer. A manufacturer CVC is always included in the code file.
- **A co-signing agent:** As described above, DOCSIS 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's CVC in the configuration file when initializing the CM's code verification process. The CM MUST verify that the co-signer's organization name in the co-signer's CVC subject field, exactly matches the co-signer's organization name previously received in the co-signer's 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 that co-signer's time-varying control values (see Section 14.3.3).

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 cable modem registers on a DOCSIS network, it 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, values for the following parameters MUST be loaded into the CM's non-volatile memory:

- CM manufacturer's organizationName;
- codeAccessStart initialization value;
- cvcAccessStart initialization value.

The CM MUST initialize the values of codeAccessStart and cvcAccessStart 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's 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 MULPI 3.0]. 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 SHOULD include 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. Regardless of whether a code upgrade is required, a CVC in the configuration file MUST be processed by the cable modem.

A configuration file may contain:

- No CVC;
- A Manufacturer's CVC only;
- A Co-Signer's CVC only;
- Both a Manufacturer's CVC and a Co-Signer's CVC.

Before the CM will enable its ability to upgrade code files from the network, it MUST receive a valid CVC in a configuration file and successfully register with the CMTS. If the CM's configuration file does not contain a valid CVC and its ability to upgrade code files has been disabled, then the CM MUST reject any information in a CVC delivered via SNMP.

When the cable modem's configuration file does not contain a co-signer’s CVC, 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's 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.[34]

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 a valid CVC, 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 a valid CVC, the CM SHOULD NOT process CVC's subsequently delivered via an SNMP MIB. If the CM configuration file does not include a valid CVC, the CM MUST NOT accept information from a CVC 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's CVC validity start time is greater than or equal to the manufacturer's cvcAccessStart value currently held in the CM if the CVC is a Manufacturer's CVC (Type 32) 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's CVC (Type 32) 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's cvcAccessStart value currently held in the CM if the CVC is a Co-signer's CVC (Type 33) 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's CVC (Type 33) and the subject organizationName is not identical to the current code co-signing agent name.
6. Verify the certificate signature against the DOCSIS root key held by the CM.
7. 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

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 it MUST reject all CVC's 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.

[34] Paragraphs 6 & 7 of this section revised per ECN SECv3.0-N-06.0322-1 on 11/17/06 by KN.

When a CM receives a CVC via SNMP it MUST:

1. Verify that the Extended Key Usage extension is in the CVC as specified in Appendix III.
2. Verify that the manufacturer's CVC validity start time is greater than the manufacturer's 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's 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 the certificate signature against the DOCSIS root key held by the CM.
6. 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.

14.3.4 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 subclause. If any of the verification checks fail, or if any subclause 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 made in any order.

1. The CM MUST verify that:
   - the value of signingTime is equal to or greater than the manufacturer's codeAccessStart value currently held in the CM;
   - the value of signingTime is equal to or greater than the manufacturer's CVC validity start time;
   - the value of signingTime is less than or equal to the manufacturer's CVC validity end time.
2. The CM MUST verify that:
   - the CVC subject organizationName is identical to the manufacturer name currently stored in the CM's memory;
   - the CVC validity start time is equal to or greater than the manufacturer's cvcAccessStart value currently held in the CM;
   - the Extended Key Usage extension in the CVC meets the requirements of Appendix III.
3. The CM MUST validate the certificate signature by using the DOCSIS Root CA public key held by the CM.
4. The CM MUST verify the manufacturer's 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.
5. If the manufacturer signature verifies and a co-signing agent signature is required:
   a) The CM MUST verify that:

(1) the co-signer's 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 CVC subject organizationName is identical to the co-signer's organization name currently stored in the CM's memory;

(2) the 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 CVC meets the requirements of Appendix III.

c) The CM MUST validate the certificate signature by using the DOCSIS Root CA public key held by the CM.

d) The CM MUST verify the co-signer's 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. Once the manufacturer's, and optionally the co-signer's, signature has been verified, the code image can be trusted and installation may proceed. Before installing the code image, 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 SHOULD be immediately discarded.
7. The CM may upgrade its software by installing the code file according to [DOCSIS MULPI 3.0]
8. 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 MULPI 3.0]
9. 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
10. 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's codeAccessStart with the [PKCS#7] signingTime value
    b) Update the current value of the co-signer's cvcAccessStart with the CVC validity start value

14.3.6 DOCSIS Interoperability

DOCSIS 3.0 cable modems MUST verify code upgrades according to this specification even when operating with a DOCSIS environment prior to DOCSIS 3.0.

DOCSIS 1.0 configuration files intended for DOCSIS 3.0 cable modems need to support the configuration file requirements that are defined in this specification in order for code upgrades to work properly for a DOCSIS 3.0 CM operating in DOCSIS 1.0 mode.

14.3.7 Error Codes

The CM MUST log the following 9 error events, when they occur, during the code verification process. DOCSIS event logging requirements and event message format are defined in [DOCSIS OSSI 3.0].

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's [PKCS#7] signingTime value is less-than the codeAccessStart value currently held in the CM
   d) The manufacturer's [PKCS#7] validity start time value is less-than the cvcAccessStart value currently held in the CM
   e) The manufacturer's CVC validity start time is less-than the cvcAccessStart value currently held in the CM
   f) The manufacturer's [PKCS#7] 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's [PKCS#7] signingTime value is less-than the codeAccessStart value currently held in the CM
   i) The co-signer's [PKCS#7] validity start time value is less-than the cvcAccessStart value currently held in the CM.
   j) The co-signer's CVC validity start time is less-than the cvcAccessStart value currently held in the CM
   k) The co-signer's [PKCS#7] signingTime value is less-than the CVC validity start time
   l) Missing or improper extended key-usage extension in the co-signer's CVC
2. Code file manufacturer CVC validation failure
3. Code file manufacturer CVS validation failure
4. Code file co-signer CVC validation failure
5. Code file co-signer CVS validation failure
6. Improper Configuration File CVC format

   Conditions:

   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

14.4 Security Considerations (Informative)

The protection afforded private keys is a critical factor in maintaining security. Users authorized to sign code, *i.e.,* manufacturers and operators who have been issued code-signing 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 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 update helps 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 [PKCS#7] structure 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 CMs 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 by a CM 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.

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 MULPI 3.0].

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 Configuration Setting

The combination of Privacy Enable configuration setting ([DOCSIS MULPI 3.0]) and the Privacy Support Modem Capability Setting ([DOCSIS MULPI 3.0]) controls whether Baseline Privacy Plus is enabled or disabled in a CM. If the operator intends to provision a CM to operate in BPI+ mode using the default BPI Configuration Parameter(s) specified in Table A–1, the corresponding Baseline Privacy Configuration subset(s) in the configuration file may be omitted. If the configuration file does not contain all the necessary BPI+ parameters, the CM MUST use the default value(s) specified in Table A–1 for the missing parameter(s). On the other hand, if the operator intends to provision a CM to operate in BPI+ mode using the BPI Configuration Parameter(s) different from the default value(s) in Table A–1, the corresponding Baseline Privacy Configuration subset(s) are present. The Baseline Privacy Configuration setting may be present if Baseline Privacy Plus is disabled. 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 MULPI 3.0] 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.

Sub-Type	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.

Sub-Type	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.

Sub-Type	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.

Sub-Type	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.

Sub-Type	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.

Sub-Type	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.

Sub-Type	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.

Sub-Type	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.

Sub-Type	Length	Minimum Value	Maximum Value
9	4	0	10

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 A–1 - 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 re-authorization	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

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

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

To be backward compatible DOCSIS 3.0 CMs MUST support the Dynamic Security Association functionality defined in this Annex when operating 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 head-end;
- 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. When in MDF-Disabled Mode (see [MULPI]), if the CM detects IGMPv2 [RFC 3376] join messages, the CM MUST trigger 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. [35]

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

[35] Paragraph 3 revised per EC SECv3.0-N-06.0322-1 on 11/17/06 by KN.

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 a 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 a 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 C–1) and as a state transition matrix (Table C–1). The CM MUST 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 CM MUST 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.

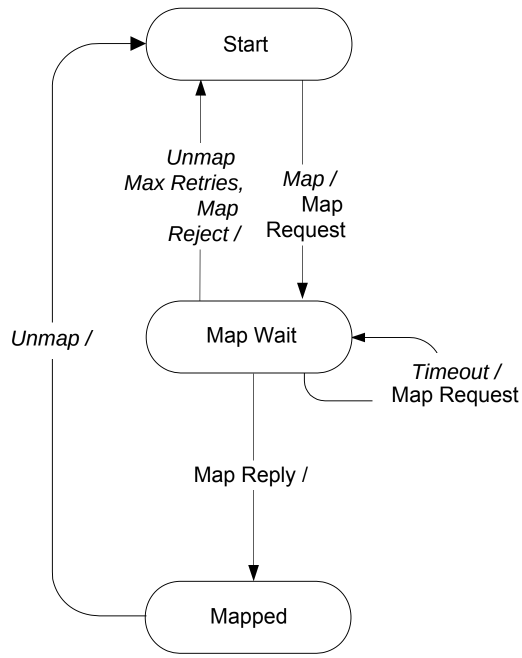

Figure C–1 - SA Mapping State Machine Flow Diagram

Table C–1 - 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]	

C.3.1 Brief Description of States

C.3.1.1 [Start]

The initial state of the FSM.

C.3.1.2 [Map Wait]

The CM has sent the CMTS a Map Request and is waiting for a response.

C.3.1.3 [Mapped]

The CM has received a Map Reply and learned the requested SA mapping.

C.3.2 Brief Description of Messages

C.3.2.1 Map Request

Sent by CM to CMTS to request a SA mapping.

C.3.2.2 Map Reply

Positive CMTS response to a Map Request; contains the requested SA mapping.

C.3.2.3 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.

C.3.3 Brief Description of Events

C.3.3.1 {Map}

Triggers the start of the SA Mapping state machine. The {Map} event is linked to a CM event not defined by this specification.

C.3.3.2 {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.

C.3.3.3 {Map Reply}

CM has received an SA Map Reply message.

C.3.3.4 {Map Reject}

CM has received an SA Map Reject message.

C.3.3.5 {Timeout}

CM has timed out waiting for a response to an outstanding SA Map Request message.

C.3.3.6 {Max Retries}

CM has sent the maximum number of retries and not received a response.

C.3.3.7 Brief Description of Parameters

All configuration parameter values are obtained from the configuration file

C.3.3.8 SA Map Wait Timeout

Timeout period between sending SA Map Request messages from SA Wait state. See Annex A.

C.3.3.9 SA Map Max Retries

This value specifies the maximum number of times that the CM may retry an SA Map Request.

C.3.4 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 BPI/BPI+ Interoperability[36]

The DOCSIS 3.0 Security specification includes enhanced Baseline Privacy Plus requirements like those defined in the DOCSIS 1.1/2.0 Baseline Privacy Plus specification [DOCSIS BPI+]. Baseline Privacy Plus is based on the original requirements of the DOCSIS 1.0 Baseline Privacy specification [SCTE 22-2]. The original architecture and design of Baseline Privacy has been maintained where possible.

The evolution to DOCSIS 3.0 Security was not intended to immediately obsolete DOCSIS 1.0, 1.1, or 2.0 systems. A DOCSIS system's transition to DOCSIS 3.0 Security compliance may be incremental. In the meantime and thereafter,

DOCSIS 1.0 Baseline Privacy, DOCSIS 1.1, or 2.0 Baseline Privacy Plus and DOCSIS 3.0 Security units may coexist within a DOCSIS system.

D.1 DOCSIS BPI/BPI+ Interoperability Requirements

BPI/BPI+ interoperability requirements are summarized in the following table. A Baseline Privacy Plus system MUST be backward compatible with Baseline Privacy according to this table. There are four unit capabilities defined here from the Baseline Privacy specification and supported by these interoperability requirements.

1. Cable Modem Termination System:
    a) CMTS BPI: Baseline Privacy with 56-bit DES, and will accept both a 768 and 1024-bit public key modulus.
    b) CMTS BPI – 40-bit: Baseline Privacy with 40-bit DES, and will accept both a 768 and 1024-bit public key modulus. DES can only operate in 40-bit mode.
2. Cable Modem:
    a) CM BPI: Baseline Privacy with 56-bit DES, and either a 768 or 1024-bit public key modulus.
    b) CM BPI – 40-bit: Baseline Privacy with 40-bit DES, and either a 768 or 1024-bit public key modulus. DES can only operate in 40-bit mode.

As defined in this specification, Baseline Privacy Plus introduces two additional unit types.

a) CMTS BPI+: Baseline Privacy Plus with 56-bit DES and 128-AES, and will accept both a 768 and 1024-bit public key modulus.
b) CM BPI+: Baseline Privacy Plus with 56-bit DES and 128-AES, and a 1024-bit public key modulus.

The CMTS and the CM negotiate the BPI/BPI+ compatible mode using the Privacy Support Modem Capability TLV (type 5.6) in the REG-REQ and REG-RSP messages. The requirements for BPI/BPI+ interoperability are:

a) A CMTS MUST accept public keys with a modulus of both 768 and 1024-bits from a CM during authorization.
b) If a CM with Baseline Privacy Plus (CM BPI+) is provisioned with a DOCSIS 1.0 style configuration file, the CM sets the Privacy Support Modem Capability TLV (type 5.6) to either BPI Support (0) or BPI+ Support (1) depending on its capability in that situation [DOCSIS MULPI].
c) When a CMTS with Baseline Privacy Plus (CMTS BPI+) receives the Privacy Support Modem Capability TLV set to BPI Support (type 5.6, value 0) or no type 5.6 TLV in the REG-REQ message from the CM, the CMTS MUST fall back into a Baseline Privacy compatible mode of operation [SCTE 22-2] for communications with that CM.

[36] Annex D added per SECv3.0-N-07.0409-1 #7 on 4/26 by KN.

d) When a CMTS with Baseline Privacy Plus is operating in a system that supports both BPI and BPI+ CMs, the TFTP server MUST include the following two kinds of configuration files:

a. Configuration file with all of the BPI parameters (type 17.1 through 17.7) for the CMs provisioned to operate in BPI mode, and

b. Configuration file with all of or a part of the BPI+ parameters for the CMs provisioned to operate in BPI+ mode.

e) When a CM with Baseline Privacy Plus (CM BPI+) receives the Privacy Support Modem Capability TLV set to BPI Support (type 5.6, value 0) or no type 5.6 TLV in the REG-RSP message from the CMTS, the CM MUST fall back into a Baseline Privacy mode of operation [ANSI/SCTE 22-2] to communicate with the CMTS.

Table D-1 BPI/BPI+ Interoperability Matrix

	CM BPI	CM BPI-40 bit	CM BPI+
CMTS BPI	Domestic BPI configuration. 768 or 1024-bit RSA modulus.	768 or 1024-bit RSA modulus. CMTS software zeros TEK bits to 40-bit standard	CM falls back into BPI mode with 1024-bit RSA modulus
CMTS BPI-40 bit	768 or 1024-bit RSA modulus. CMTS software zeros TEK bits to 40-bit standard.	768 or 1024-bit RSA modulus. All 40-bit compatibility handled by MAC chips.	CM falls back into BPI mode with 1024-bit RSA modulus. CMTS software zeros TEK bits to 40-bit standard
CMTS BPI+	CMTS falls back into BPI mode. 768 or 1024-bit RSA modulus.	768 or 1024-bit RSA modulus. CMTS software zeros TEK bits to 40-bit standard	Full BPI+ mode or BPI mode depending on configuration file and CMTS setting. 1024-bit RSA modulus.

D.2 BPI 40-bit DES Export Mode Considerations

The DOCSIS 3.0 Security specification is backward compatible with the 40-bit DES export mode of Baseline Privacy. The burden of compliance is placed on the CMTS. Not all DOCSIS equipment vendors will ever have the need to operate in a system with 40-bit DES capable BPI units. Therefore, compliance is up to the individual CMTS manufacturer. A CMTS SHOULD support backward compatibility to 40-bit DES Baseline Privacy. If it does, it MUST do so according to section 11 of this specification.

a) When a CMTS is sending or receiving encrypted data between itself and a CM that uses 40-bit DES, the CMTS MUST zero the appropriate bits of its TEKs before encrypting or decrypting corresponding traffic data. The appropriate bits of the TEK MUST be zeroed according to the 40-bit TEK requirement of Baseline Privacy.

b) When encrypted traffic is to be passed between a CMTS with only 40-bit DES capability and a CM with a 56-bit DES capability, the CMTS MUST provide a 40-bit compliant TEK in the Key Reply Message to the CM.

The method a CMTS uses to recognize which CMs in a system are capable of 56-bit DES or only 40- bit DES, is left up to the individual system operator and CMTS vendor to accomplish in the manner that best fits their situation. One method for obtaining this information would be from the CM vendors, based on CM serial numbers, MAC address, manufacture dates, or some other device tracking mechanism. Once collected, the information would be incorporated into the CMTS database of information stored on each CM.

An alternative method for obtaining this information is with a DOCSIS BPI MIB defined for this purpose.

D.3 System Operation

D.3.1 D.3.1 CMTS with BPI Capability

A CMTS with BPI capability will always provision CMs using DOCSIS 1.0 style TFTP configuration files and BPI configuration settings. Both the BPI and BPI+ CMs will receive the BPI settings and each CM will only attempt to register as a DOCSIS 1.0 CM with BPI capability. If a CM returns a Modem Capability of BPI+ in the registration request, the CMTS will respond with this capability removed and force the CM to BPI compatibility.

D.3.2 CMTS with BPI+ Capability

A CMTS with BPI+ capability MUST be capable of operating in both BPI and BPI+ compatible modes and to adjust according to the capability of each client CM. When the CMTS has BPI+ capability and the system simultaneously supports BPI and BPI+ CMs, both DOCSIS 1.0 and DOCSIS 1.1, 2.0, or 3.0 configuration files MUST be available to deliver the BPI+ and BPI configuration settings to the appropriate CMs. A BPI capable CM will receive a DOCSIS 1.0 configuration file with BPI settings. It will then register with BPI Modem Capability.

Appendix I Example Messages, Certificates, PDUs and Code File (Informative)

This appendix presents detailed examples that may be useful to implementors 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 shall take precedence.

I.1 Notation

In the examples here, packets are represented as a stream of octets, each octet in hexidecimal 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 hexidecimal 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 hexidecimal notation using an "0x" prefix or in decimal notation with no prefix. For example, the hexidecimal 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 02 94	Auth Info header
11 02 91	CA Certificate header
30 82 02 8d 30 82 01 f6 . . . 81 87 19 61 72 20 19 1e	CA Certificate

The code field has value 0x0c, which identifies this as an Authentication Info message. The Length field has a value of 0x294 (660), 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 CA Certificate details

The fields of the CA Certificate in the Authorization Info message are as follows:

30 82 02 8d	Certificate header
30 82 01 f6	TbsCertificate header
a0 03 02 01 02	version
02 08 01 02 03 04 05 06 07 08	serial number
30 0d 06 09 2a 86 48 86 f7 0d 01 01 05 05 00	signature
30 81 88	issuer header
31 0b 30 09 06 03 55 04 06 13 02 55 53	country name
31 0f 30 0d 06 03 55 04 0a 13 06 4e 6f 72 74 65 6c	Organization name
31 0f 30 0d 06 03 55 04 0b 13 06 44 4f 43 53 49 53	Organizational unit name
31 1f 30 1d 06 03 55 04 0b 13 16 42 75 69 6c 64 69 6e 67 20 31 2c 20 41 6e 64 6f 76 65 72 20 4d 41	Organizational unit name
31 36 30 34 06 03 55 04 03 13 2d 4e 6f 72 74 65 6c 20 43 61 62 6c 65 20 4d 6f 64 65 6d 20 52 6f 6f 74 20 43 65 72 74 69 66 69 63 61 74 65 20 41 75 74 68 6f 72 69 74 79	common name
30 1e	Validity header
17 0d 39 39 30 31 32 30 31 36 30 35 30 30 5a	not before
17 0d 34 39 31 32 33 31 32 33 35 39 35 35 5a	not after
30 81 88	Subject header
31 0b 30 09 06 03 55 04 06 13 02 55 53	country name
31 0f 30 0d 06 03 55 04 0a 13 06 4e 6f 72 74 65 6c	Organization name
31 0f 30 0d 06 03 55 04 0b 13 06 44 4f 43 53 49 53	Organizational unit name
31 1f 30 1d 06 03 55 04 0b 13 16 42 75 69 6c 64 69 6e 67 20 31 2c 20 41 6e 64 6f 76 65 72 20 4d 41	Organizational unit name
31 36 30 34 06 03 55 04 03 13 2d 4e 6f 72 74 65 6c 20 43 61 62 6c 65 20 4d 6f 64 65 6d 20 52 6f 6f 74 20 43 65 72 74 69 66 69 63 61 74 65 20 41 75 74 68 6f 72 69 74 79	common name

30 81 9f	subject public key info header
30 0d 06 09 2a 86 48 86 f7 0d 01 01 01 05 00	public key algorithm type
03 81 8d 00 30 81 89	public key header
02 81 81 00 af d1 86 c8 17 45 02 bc e5 59 b4 15 ac 95 87 7b 89 f5 8b f8 3b 8a 8b ef 67 cf 9e 00 47 d5 f1 06 42 55 36 a1 d1 8c dc cb 81 bb 31 8d 35 f7 6d 11 a0 91 9b 31 3d b9 71 38 46 15 c8 81 c4 51 06 7b d7 8a 70 be c1 28 0d 78 80 3c 44 a6 5e 35 5f 6e 46 2f 80 41 28 78 63 6c 86 cc d0 b3 58 ca bc 07 d5 19 3e 8a a2 1c 7e ff 0d 16 2b 0f bd a5 5e 60 93 64 09 80 24 76 ed e4 a9 e3 81 26 0c de 8a 89	public key modulus
02 03 01 00 01	public key exponent
30 0d 06 09 2a 86 48 86 f7 0d 01 01 05 05 00	signature algorithm
03 81 81 00 81 4d db 31 e2 31 d2 6c f5 21 29 93 4a ce cb 6c fb 8b fc 3d ef 4b e8 4a 8a db f7 d8 e3 70 1d 3c ff ba 71 70 c4 82 24 9f 12 b5 d4 3e 3a 4d 20 64 2f ab 8b 05 27 9a 34 24 33 24 d4 7e bc 41 07 34 7a a6 51 12 29 55 e7 9b 5b e5 6b 79 bb 31 04 2f d1 c6 d3 7f 32 a2 b5 cc 99 23 09 97 1a 21 44 fa 25 3b f4 4b d6 00 cf e9 1b a9 be 9b 88 f8 90 fd 59 77 80 41 7d cb ca bf 81 87 19 61 72 20 19 1e	signature value

Some of the fields in this example are identical in all CA certificates. These fields are:

- version: v3
- signature: SHA-1 with RSA, NULL parameters
- subject first organizational unit name: "DOCSIS"
- public key algorithm type: RSA encryption, NULL parameters
- public key exponent: 3-octet integer, value 0x10001
- signature algorithm: SHA-1 with RSA, NULL parameters

This is an example of a self-signed CA certificate. The issuer name and the subject names are identical. In this example, the matching name fields are:

- country name: "US"
- organization name: "Nortel"
- first organizational unit name: "DOCSIS"
- second organizational unit name: "Building 1, Andover MA"
- common name: "Nortel Cable Modem Root Certificate Authority"

The other fields are example values. Some of these are:

- serial number: INTEGER of 8 octets, value 0x0102030405060708 (other CA certificates may use a different length)

- not before: 1999-01-20 16:05:00 UTC
- not after: 2049-12-31 23:59:55 UTC
- public key modulus: INTEGER of 1024 bits, value 0x00afd1...8a89 (other CA certificates may use an integer of a different length)
- signature value: A BIT STRING representing the INTEGER value 0x00814d...191e, (Other CA certificates may use a BIT STRING of a different length. 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 72 03 40	Auth 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 00 00 ca	Manufacturer ID
03 00 06 00 00 ca 01 04 01	MAC Address
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	RSA Public Key
12 02 7a	CM Certificate header
30 82 02 76 30 82 01 df . . . 19 c9 f1 dc 30 b8 d3 d5	CM Certificate
13 00 0b	Security Capabilities header
15 00 04 01 00 02 00	Cryptographic Suite List
16 00 01 01	BPI Version
0c 00 02 22 60	SAID

The Code field has a value of 0x04, which identifies this as an Authorization Request packet. The Identifier field has value 0x72; this is an example value. The Length field has value 0x0340 (832), 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 section 2.2 of [RSA2]. The modulus is a 1024-bit integer encoded in 0x81 (129) octets. In this example, the value of the encoded modulus is:

0x00e0e06c8d ... caeed631.

0x00 is the most significant octet of the encoded modulus and 0x31 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, two Cryptographic Suites are listed: 56-bit DES with no authentication, and 40-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 0x2260.

I.3.1 CM Certificate details

The fields of the CM Certificate in the Authorization Info message are as follows:

30 82 02 76	certificate header
30 82 01 df	tbsCertificate header
a0 03 02 01 02	version
02 08 01 01 01 01 01 01 01 01	serial number
30 0d 06 09 2a 86 48 86 f7 0d 01 01 05 05 00	signature
30 81 88	issuer header
31 0b 30 09 06 03 55 04 06 13 02 55 53	country name
31 0f 30 0d 06 03 55 04 0a 13 06 4e 6f 72 74 65 6c	organization name
31 0f 30 0d 06 03 55 04 0b 13 06 44 4f 43 53 49 53	organizational unit name
31 1f 30 1d 06 03 55 04 0b 13 16 42 75 69 6c 64 69 6e 67 20 31 2c 20 41 6e 64 6f 76 65 72 20 4d 41	organizational unit name
31 36 30 34 06 03 55 04 03 13 2d 4e 6f 72 74 65 6c 20 43 61 62 6c 65 20 4d 6f 64 65 6d 20 52 6f 6f 74 20 43 65 72 74 69 66 69 63 61 74 65 20 41 75 74 68 6f 72 69 74 79	common name
30 1e	validity header
17 0d 39 39 30 33 32 33 31 36 35 38 33 34 5a	not before
17 0d 34 39 31 32 33 31 32 33 35 39 35 30 5a	not after
30 72	subject header
31 0b 30 09 06 03 55 04 06 13 02 55 53	country name
31 0f 30 0d 06 03 55 04 0a 13 06 4e 6f 72 74 65 6c	organization name
31 1f 30 1d 06 03 55 04 0b 13 16 42 75 69 6c 64 69 6e 67 20 31 2c 20 41 6e 64 6f 76 65 72 20 4d 41	organizational unit name
31 15 30 13 06 03 55 04 03 13 0c 30 30 30 30 30 30 31 32 33 34 35 36	common name (serial number)
31 1a 30 18 06 03 55 04 03 13 11 30 30 3a 30 30 3a 43 41 3a 30 31 3a 30 34 3a 30 31	common name (MAC address)
30 81 9f	subject public

	key info header
30 0d 06 09 2a 86 48 86 f7 0d 01 01 01 05 00	public key algorithm type
03 81 8d 00 30 81 89	public key header
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	public key modulus
02 03 01 00 01	public key exponent
30 0d 06 09 2a 86 48 86 f7 0d 01 01 05 05 00	signature algorithm
03 81 81 00 19 b0 2b e5 2c 37 4a af 34 cb c9 59 62 68 88 05 8a 91 5b d4 c6 fa 2e 19 ab 98 42 33 68 9d fc e4 76 23 84 8d 4a be ff bf 34 cf e0 fb 93 96 01 8b 89 d9 86 42 5e cf 6d e6 68 2e 44 99 56 6a cc f1 2c b9 5b 30 21 08 22 f5 11 b1 38 ba 6e b5 62 f0 3a dc f1 2e c4 61 95 2f 16 c8 27 63 b6 e8 69 a6 1c e1 4f 1a 8c 65 cb 57 5e 13 ce db 7f 27 f9 c1 6e bf 2f 75 77 9e a9 87 19 c9 f1 dc 30 b8 d3 d5	signature value

Some of the fields in this example are identical for all CM Certificates. These fields are:

- version: v3
- signature: SHA-1 withRSA, NULL parameters
- issuer first organizational unit name: "DOCSIS"
- public key algorithm type: RSA encryption, NULL parameters
- public key exponent: 3-octet integer, value 0x10001
- signature algorithm: SHA-1 withRSA, NULL parameters

The issuer name of the CM certificate matches the subject name of the CA certificate. In this example, the matching issuer-name fields are:

- country name: "US"
- organization name: "Nortel"
- first organizational unit name: "DOCSIS"
- second organizational unit name: "Building 1, Andover MA"
- common name: "Nortel Cable Modem Root Certificate Authority"

The other fields are example values. Some of these are:

- serial number: integer of 8 octets, value 0x0101010101010101 (other CM certificates may use a different length)
- not before: 1999-03-23 16:58:34 UTC
- not after: 2049-12-31 23:59:50 UTC
- subject country name: "US"

- subject organization name: "Nortel"
- subject organizational unit name: "Building 1, Andover MA"
- subject first common name (serial number): "000000123456" (Other CM certificates may use a different length string. The value matches the Serial Number attribute of the Authorization Request message.)
- subject second common name (MAC address): "00:00:CA:01:04:01" (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 1024 bits, encoded as 0x00e0e0...d631 (Other CM certificates may use a modulus of length 768 or 1024 bits.)
- signature value: BIT STRING representing the integer value 0x19b0...d3d5 (Other CM certificates may use a BIT STRING of a different length that encodes a modulus of 1024 to 2048 bits. 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 72 00 9f	Auth Reply header
07 00 80 a2 cb ad c8 34 27 71 47 06 d5 10 0c 07 94 90 bf e6 44 1b 0c 90 0d b4 ed 9c 39 aa 05 a0 c1 ef 54 4b cc fb 3a 7a 22 81 c0 dc c6 6e 39 a4 91 1c ba bf b0 ed 47 10 f2 f4 13 f9 09 33 c6 ae a3 45 67 c8 38 0f c3 9a 12 be d5 27 27 39 77 fb 98 03 39 50 39 99 f5 b6 ad b5 85 f9 16 d0 ff c6 2a ff 9f 38 73 6f 35 44 21 ad 9e e1 a5 91 4d 34 06 1d bb c9 b6 8f 8a 17 9e be c6 c9 40 eb 81 f0 62 d8 18	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 0x72, matching the Identifier field of the Authorization Request. The Length field has value 0x009f (159), which is the number of octets that follow the Length field.

The first attribute is the Authorization Key. The attribute contains an authorization key which has been RSA-encrypted using the public key in the Authorization Request message. The RSA-encrypted authorization key is an integer made up of 0x80 (128) octets. In this example, the value of the RSA-encrypted authorization key is `0xa2cbadc8 ... f062d818`.

0xa2 is the most significant octet of the RSA-encrypted authorization key and 0x18 is the least significant octet. Details of the encryption calculation are given below.

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.

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.4.1 RSA encryption details

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]. This section gives details of the scheme as applied to this example. The scheme makes use of a mask-generating function (MGF) which is based on hashing; details are given in a later section.

The authorization key is padded into a 107-octet data block DB:

```
DB =
da 39 a3 ee 5e 6b 4b 0d 32 55 bf ef 95 60 18 90 af d8 07 09 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01
42 85 27 ff c4 12 72 8e 61 84 de c9 20 b6 e0 64 f0 bc ob 75
```

To form DB, the authorization key is prefaced with an octet of value 1, and the result is placed in the last 21 octets of the block. The first 20 octets of the block are the result of performing a hash operation on a zero-length string; these 20 octets have the same value in every Authorization Reply and are not unique to this example. The remaining 66 octets of the block are set to 0.

The CMTS generates a string of 20 octets called the SEED. The SEED is different for each Authorization Reply. In this example, the SEED has value:

```
SEED =
ad 9c af 8d f8 26 fe af b5 df fd 95 de 7e 97 cc e9 4b 6d 6d
```

The SEED is input to the MGF to generate DB_MASK, a block of 107 octets:

```
DB_MASK =
de 10 c9 59 41 c9 ea 72 a4 35 68 79 d2 53 85 db 13 7b a6 3b 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 c 3f 6e
ac 99 3a ae 14 3e 9a 8e df 3c 36 79 58 b2 fa 13 72 58 4c ca 04 a1 af c7 c4 62
```

DB and DB_MASK are XORed to produce MASKED_DB, which has 107 octets:

```
MASKED_DB =
```

```
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
```

MASKED_DB is input to the MGF to generate SEED_MASK, a block of 20 octets:

```
SEED_MASK =
b4 b6 f1 bf a6 b3 a1 7e 95 82 d3 b8 93 71 b6 7f 45 31 9e 82
```

SEED and SEED_MASK are XORed to produce MASKED_SEED, which has 20 octets:

```
MASKED_SEED =
19 2a 5e 32 5e 95 5f d1 20 5d 2e 2d 4d 0f 21 b3 ac 7a f3 ef
```

MASKED_SEED and MASKED_DB are concatenated, and the result is prefaced with a single octet of value 0. This results in a 128-octet block called EM:

```
EM =
00 19 2a 5e 32 5e 95 5f d1 20 5d 2e 2d 4d 0f 21 b3 ac 7a f3 ef 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
```

To perform RSA encryption, EM is interpreted as the integer value:

0x00192a5e32 ... 5f7bcf17.

0x00 is the most significant octet and 0x17 is the least significant octet.

The RSAencryption is performed as the operation Y = ME mod N, where:

M is the integer value of the block EM (0x00192a5e32 ... 5f7bcf17); E is the integer value of the exponent of the public key (0x010001); N is the integer value of the modulus of the public key (oxe0e06c8d ... caeed631); Y is the integer value of the encrypted authorization key (0xa2cbadc8 ... f062d818).

I.4.2 RSA decryption details

Table I–1 lists the private-key parameters that match the RSA public key in the example Authorization Request message.

Table I–1 - Private Key Parameters

Parameter	Property	Value
D (private exponent)	M^{DE} mod N = M	6b 1f 1d 36 ec 77 7b 15 a9 c6 30 27 71 ae 92 62 3a 9f 67 47 d8 00 9d ca a0 0b f9 a6 0d be 54 3d 5a 6e be 25 25 bc d9 67 da 7b 80 5f a1 c6 75 67 dd 84 ba 4b 16 26 ba e9 fd 61 ab cd 49 e0 18 47 37 9f 56 08 2d d9 16 81 ff 7d d0 7e 01 8f d4 84 d3 e8 eb 27 48 c3 6c dc a9 01 b7 e5 24 28 d1 6c 67 03 a7 63 fb fa 79 d8 08 6a e1 de 3d 12 7a 36 20 25 01 d1 08 11 0c cd 80 44 3c fd c5 c4 db d1
P (prime factor)	N = PQ	f1 6b dd 2f dd d8 df 80 30 e6 9c d3 4e 46 5e 9f 42 62 b1 66 86 57 1b ca 87 9c cf fd 1c b6 26 76 95 35 bf 0b fb 51 af 0f 46 1c 5e cb 82 a0 83 bf 46 c9 3b d6 4e 7a 5d bf 03 05 69 27 31 6d 65 bd

Parameter	Property	Value
Q (prime factor)	N = PQ	ee 74 cb a3 d0 90 2d 8a e9 e7 10 dd b4 65 2e 91 22 09 52 72 ab bd 32 31 4e d7 d0 2b 4b 13 57 20 6b f9 a4 57 b1 47 59 67 86 a6 8c 2c c1 f3 8b ba 8a 6b b1 62 5d 43 5a 71 db d0 33 43 97 99 17 85
D_p (CRT exponent)	D_p = D mod (P - 1)	a6 35 dc d2 57 aa 38 35 c9 74 fc 03 7e a0 74 04 b1 6f c1 33 14 ca 64 17 cb c5 ea 6c 18 98 4f 62 d4 d7 6b f0 93 d6 68 ef db 15 2d 2e 6f 80 93 33 dd 48 2e 2a 1d 5d a1 ad 20 27 59 7d e2 49 af 01
D_q (CRT exponent)	D_q = D mod (Q - 1)	cf f1 9c 30 33 cd b7 59 7f 96 57 f7 ee bb 99 bb 48 a2 36 7a f7 57 1a f1 32 df 32 92 be 7a 94 2d 1a db ed bb e7 45 e0 2a 4e 9a e8 7c 93 7a 4e 2c 93 4f 4c b6 09 bc 95 9f da df 9a 04 e4 ab c5 7d
U_p (CRT constant)	PU_p mod Q = 1	08 17 0c 11 bc aa 2f 96 80 8b 31 95 6d 2e b8 3c ee 2e 05 88 ab 9e fc 53 24 c4 04 b8 7e 1d 01 db 2d f2 2c 06 b0 cd 04 6b 1c 14 d8 d0 4f c9 a0 ae 1b c9 80 88 be 42 0a 52 4a ef 62 3c 8b dd c5 37

Each value in Table I–1 represents the octets of an integer, with the most significant octet shown first. For example, the private exponent D has the integer value:

0x6b1f1d36 ... c5c4dbd1.

The CM can decrypt the authorization key with or without using the Chinese Remainder Theorem (CRT). Decryption using the CRT is more complicated, but it may be faster than performing decryption without it.

To decrypt without using the CRT, the CM performs the operation M = YD mod N. D is the private exponent in the table, and Y and N are as described in the preceding section. The resulting value matches the value of M in the preceding section, that is, it is the integer value the block EM formed by the CMTS. The CM decodes the authorization key from EM by inverting the procedure used by the CMTS to form EM, as described in [RSA3].

To decrypt using the CRT, the CM first computes two intermediate quantities:

$A = Y^{Dp} \bmod P;$

$B = Y^{Dq} \bmod Q.$

P and Q are the prime factors of the modulus, and Dp and Dq are private exponents related to these factors, all with values shown in the table. The CM computes the value of M as:

$M = A + ((B - A)Up \bmod Q)P$

Up is a constant derived from the prime factors, with the value as shown in the table. The resulting value of M matches the value that would be computed using the operation M = YD mod N.

I.4.3 Hashing details

The authorization key is hashed using the SHA-1 algorithm [FIPS-180-2] 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 appendix B of [FIPS-180-2]:

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 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 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 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 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 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 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 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 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

Mode	CBC
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 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 test versions of the Manufacturer and Co-signer CVCs. The corresponding private keys of these CVCs are as follows:

Test Manufacturer CVC Private Key	30 82 04 be 02 01 00 30 0d 06 09 2a 86 48 86 f7 0d 01 01 01 05 00 04 82 04 a8 30 82 04 a4 02 01 00 02 82 01 01 00 a2 05 d2 10 02 f3 47 b4 dd 2e 2a 9b 05 b3 86 38 a4 91 6d ad 39 5a da 0b 48 55 9b 94 dd 4e 9d 8b 98 c6

	18 6d e8 de 1e 81 db ce 8f 3c 6d 02 81 fa 1e 46 f1 93 30 c2 79 bb ba f6 50 b2 db 10 b2 13 c0 bd a5 4d 33 5f 43 0d 9f a0 de 9b 8b 71 b5 ff 1f 48 e1 06 38 71 b8 df 3c 93 c6 f8 eb 90 a6 d2 a4 aa 7b 4e f1 2e 5c 11 07 88 1d 13 f8 60 d8 e2 3b e5 c7 7b 30 39 74 75 94 e3 73 19 92 b6 81 2f b7 db 85 be 6a 8e 54 e9 dd 67 8f a7 a3 4c d6 22 50 a8 0d 69 41 98 d5 a3 af 51 8d 48 44 f4 35 9d 5d 7d a3 d1 62 e4 01 3f 62 05 e1 89 57 5e 5b 90 09 ff 83 7b b6 f2 0d 32 72 47 9e e2 d6 1a 15 c0 97 53 9a a8 9f 24 5c 88 cf bb 98 b4 da 01 73 33 31 ea aa 8d e2 01 9f 20 3e e1 67 dd 36 2e 3c 83 28 5d 28 bb 6a d9 30 94 37 e6 d8 86 42 32 c7 9b 36 68 65 3b 3f 40 f3 87 1a 31 92 90 93 a3 21 b1 02 03 01 00 01 02 82 01 00 05 c4 89 46 d2 a4 10 70 6a d3 42 e7 03 db f1 32 41 01 80 53 03 50 ad a4 6b 2d c8 fd 12 a7 c2 c9 fe 65 46 e7 5a bb d5 80 2f ef f9 7f 60 2e 0f ac 74 b9 77 ce 12 95 49 ba 1a 1c bc a0 b1 55 b5 d1 28 32 58 19 18 74 12 b2 00 75 bc 96 f2 ee d8 58 bd a7 5d 83 58 2d 0c 34 be 36 50 85 f9 f4 cb 4b 82 5b 23 4a 9b 21 58 5b ae d5 f4 9b 0a 63 80 35 05 86 d7 84 16 22 65 77 ee 15 ee ba 01 54 07 e3 39 79 94 1f fa 34 fe 0c cd 58 dc 01 5b 55 39 ed e0 85 05 7e 94 b7 54 97 44 3b 12 be c8 36 7c bc 7a 9a e5 0d c1 27 d3 84 bb cf 5a 72 7e 55 0b 62 63 21 95 ff 7c 21 b2 c4 7a 25 b0 95 59 19 07 9f c2 4d 61 93 1a 00 46 69 6c 3e 80 61 43 0b 85 98 fe ba fc 69 df 3d 06 15 20 b5 65 38 9a d2 e0 d8 92 6a 08 42 fe d2 2c 38 ec c6 9c cb d2 cf 6a 88 11 d1 84 77 95 f9 f7 d7 7e 97 fb 34 cf b3 0c 31 02 81 81 00 d1 eb d4 44 28 f0 b2 c1 1e 59 6b 45 61 b5 4c 74 e4 73 6c c3 e6 f6 d6 c2 96 cc e6 cc 3e 27 bc 21 cb 66 28 b3 53 33 b4 df 2b 93 ee 1c ed 5f ce 18 c6 85 23 28 9b aa cd e4 34 80 92 13 1d d8 34 19 6c 3a 6e 81 7e 86 02 09 9f 4e a7 ce 52 c5 ae 93 f4 d9 2f 11 bf 9a 5f 1c 1c 63 48 40 58 02 ad 92 53 c4 da 97 86 98 30 d1 3a 16 25 58 d9 e2 af 94 65 14 71 e3 b8 be 06 0f a9 dc 3f bf a9 1c 2d a5 02 81 81 00 c5 96 6a c0 1e 2d 21 cc 33 4b e1 a4 ef 57 18 f9 32 63 97 24 9e 20 5e ef ed cb c2 0a e6 4a 02 de 4a 49 8e 23 a2 71 91 8d d3 05 2e 25 55 4e 5b 7d ca 72 e6 84 40 58 d0 ad b7 3a f4 56 64 b6 1d b2 ce cc d0 5d 99 9c 00 fd 86 ae 7d b5 c2 f3 46 98 66 38 17 6d 3f d2 87 df 65 eb 50 00 ef c8 9e 40 e7 9f 24 e0 f2 80 16 cb 54 c5 aa 05 9e 62 29 6b 9f 01 d7 59 73 af d3 bd 2d e4 fb 62 51 94 3e 1d 02 81 81 00 bf f8 97 c4 c1 5f e5 48 e0 55 4b d9 46 52 83 19 fe f5 63 88 ff c5 ca 7e 67 63 f8 c8 dd 2a 7f 4c b4 72 b7 53 cf 09 e1 25 99 f7 8d 70 dc cf 99 6d 64 d6 ca 61 4f 82 73 09 77 ce 00 48 05 29 fb f3 27 db 83 c5 5e 8c 74 7e 92 36 5f 15 3a f2 d7 05 2d 8c 14 14 7a 11 55 5b 22 c6 d4 83 2b 39 3f bc 2a 69 42 b2 01 7f 5a 5d 6c 6e 66 37 0c 02 d7 28 b1 0b 72 5f 9b 4a 03 c6 fd 42 92 f5 d6 54 13 95 02 81 81 00 8c 03 12 92 20 d0 0a 6b 10 ce 6c 8e db aa f4 5f 61 7d 80 d4 c9 d8 56 6d de 2b cc cc bc 73 25 a5 2f 2f 00 31 f0 22 59 ea 0d e6 55 d4 ce a6 3f de db 5e a0 4a 86 ee 68 b1 30 6d ee 46 6a 07 42 f4 39 e6 98 a0 3b 01 79 12 2e 25 e7 9e ad 0e 86 af e7 7f 47 da 7a a7 4c f0 d8 df ee 49 3b a1 05 c3 28 5e 75 5b 91 7b 82 ab 82 de ae a4 5c 9f 62 8a 88 9a 5d 67 5d ba f2 67 d3 e5 b6 dc ec c2 07 61 02 81 80 77 fc 72 ba d8 5d 64 25 16 71 01 b6 34 4c 24 26 33 23 8f 7e 22 e0 2e ae 7e e8 a6 b3 56 2d 9a 53 db 83 2b 1d 32 d3 9e 0c 66 71 3e 27 65 6b d9 58 7c 10 82 c8 83 f7 2e 4e 34 16 a6 ed 03 74 4e c4 7c 6f bd 76 1e 0b 23 31 89 e1 b5 a4 30 60 14 f4 1f 02 96 5f 61 42 b1 4f cd 7a 66 c3 d4 25 cc 66 c4 f9 f9 d5 33 b9 f5 ca 5c d1 ec 42 1e 4f 7a 4f 84 de a5 2b a4 ca 4b 4e b3 c2 ae 46 f9 cf a3 0a
Test Co-Signer CVC Private Key	30 82 04 bd 02 01 00 30 0d 06 09 2a 86 48 86 f7 0d 01 01 01 05 00 04 82 04 a7 30 82 04 a3 02 01 00 02 82 01

	01 00 ba d2 ed 99 85 b8 c5 6f d8 c2 20 40 20 19 ff ef ec 79 16 4f 21 93 47 dd 03 b2 41 d0 2e 85 a8 46 8e 5d 83 e7 10 43 a8 bd c2 00 cd a4 9b a7 8b 46 91 ea 64 2e 56 79 b2 84 7f ff 0e 91 9a 12 fc 99 af 92 f3 70 48 a8 50 ab 0d b9 41 65 48 f5 c8 78 23 0d 6b 57 60 ec 64 4d 10 7f 05 49 1e ec 8c 86 fd 5e 91 f5 34 3e dc 21 3e 99 6c e8 68 c3 9a f6 38 30 c2 ac 0e b4 5a 50 df d4 8c 5d 08 8d 8f 97 91 e3 6f 7b 6d 1e 16 b3 38 29 8b ef 4e b2 ea 02 f0 4d 28 f6 7a 85 8d 8c 04 ed 73 d6 f7 fd 01 74 a8 6e 82 d9 e7 b2 15 9d 9e c2 83 cb bf 77 e5 69 18 d5 4a e5 91 17 08 40 61 9d a9 77 37 6b 0f e2 3a 7d 35 d1 1f be 3a ea ed e6 fd de e0 f2 7f 79 3b a1 f0 26 8e 69 97 44 b9 e3 4d 24 8f 1e 26 1d 0c fa bc cd 2b e4 e6 f0 54 55 94 e5 6b 26 dc 30 5f fc 3e 1e cd 2d 28 36 58 bc f3 f8 f0 f5 4a e7 02 03 01 00 01 02 82 01 00 5d e1 82 de 63 f1 2e dd fe 99 c6 a5 12 bc 9d fd 4d e2 a7 50 4a 79 fd 61 d5 ea ac e1 d5 8e fa b9 c1 ee c2 4a c9 c4 86 36 1d 18 54 3e 84 ad 93 f2 79 7e c4 aa f8 6f 9d 42 b2 23 54 89 dc 57 f0 31 b6 a5 ec c9 1e e9 dd e7 a7 34 d1 e4 f8 dd f1 c8 55 2c f8 24 e7 a2 c2 e0 6b 0a b9 78 3e 1e f6 9b 1b 15 7b d9 31 50 e9 43 97 76 77 e0 ee 4c 65 41 97 89 01 3a 7f 05 28 61 7d a9 0c 83 ef b9 5b 4f 1e 56 9c b8 34 a2 b3 9b 0f 18 ee e8 ad 15 2c 02 15 56 40 b6 0a b0 8f a9 7c 27 d6 8a a9 ea 20 e3 91 b8 12 7b bd 96 0d 2c 7c 9b 3b e4 5d f0 26 4f 82 9b 35 15 9b 85 0a 66 bd 68 fa f6 f2 2c f1 f7 aa d9 a9 fb 55 33 fc 1d 89 4d 97 22 c8 dd a7 bb c0 d5 3e b5 10 91 b6 87 c8 8d 2e fc 5b a4 68 2b b8 3c 1b ee 79 1d 43 08 e7 10 15 cd 2d de 47 62 0b c9 8a 88 35 5e 33 0e a4 ac 0d 50 8a f0 de c1 02 81 81 00 e4 10 4a 86 16 be 53 0c 96 7d 7d 0a 04 27 d1 77 be 40 77 fb b3 5a 9f c9 1a 97 a6 7c a9 db ca 4d 1a a3 ef 11 9c 34 e3 70 05 21 14 c9 c7 34 1b 99 11 2d 9e 56 c9 55 a3 11 bd e2 6b a6 79 08 e1 6c 87 6a d7 6a ec a4 fa 32 27 cb 8c 52 b0 c0 28 2f 37 9a 4c ac d8 fd a5 10 2c 99 31 8a 0c 5d 9c db b6 25 33 cd 12 71 79 1d 8c d9 5f 4c 40 ad 00 52 b5 c2 cf cf 99 0c f9 0e 45 e7 e5 2c c7 b9 ee a1 02 81 81 00 d1 b5 6d 26 a4 1b be d9 4c f3 49 11 16 1a fe dd c8 e5 91 51 01 36 e5 a1 4e 06 a5 fe 0a 27 41 c2 7c db 20 0b dc 41 00 40 e1 37 b6 4a e0 9f 3b 70 3d 4d 15 f3 4a fa f1 28 a3 d7 a8 6f 73 84 e7 6a 0e 71 eb 90 3c 8a 72 2b fc c0 bf 94 d9 07 d2 e5 dc 01 3c a5 a4 3c 8b 1c 74 30 87 80 57 3b 1b a0 67 7a 6c 1e 2a 99 d9 ff 02 46 73 2b 44 cd 02 2d a0 ce 65 a4 df e3 f3 f2 b6 29 20 74 36 17 f4 87 02 81 80 6a fa d1 5a 70 d4 ba 74 4f 2b 72 39 fe 95 07 07 1a 71 77 56 d4 5d 10 52 ef 6c 6f a6 4e af 51 08 36 14 ec 11 e0 aa 56 7d b7 8a 3b 0c 23 ab 67 be b7 76 24 ae 25 27 4c a5 07 e3 6c 07 dd 02 b3 6b 8b 62 2c c6 75 57 36 1d cc d9 08 6b c9 89 79 d0 1c 4e fe e4 9d ac ed 7e d6 cc cd 96 2a 12 c7 7f b0 c3 66 b7 2d 8d 98 f8 a9 97 55 c2 99 1c 0d e8 be 66 31 c0 f9 d4 42 07 d5 f6 df 63 d0 14 81 c1 02 81 80 18 a1 17 35 af 96 76 9e ac d3 f1 e3 e8 d5 b8 d3 31 72 0c d4 17 b1 6b dc ba fe 3a e6 90 28 b6 27 72 64 85 e9 fe af 3f a9 5f 0a 1d 05 c4 45 cf 44 4a 5e d0 52 27 9f 3b 4d 8b 98 9b 76 a3 79 9b 78 30 44 1e e7 6c c4 d1 cd ea bc 74 83 b5 cd 9e 6e 52 9d 6e 93 01 ab fe a6 93 da d3 3f 36 7f 81 aa 20 1d 17 af 9f 4f 3c 74 1f bf 48 6d 6f 82 28 5c 55 83 88 1a 8a 15 0a a4 99 d2 50 b6 75 57 f0 e7 02 81 81 00 87 21 e9 99 f4 5a 16 78 77 48 2f a7 8e 55 80 9d e4 b6 a7 c0 20 7b f7 3b 31 96 59 60 e8 20 e3 2d 22 87 16 d0 88 1f 26 a7 0e ad d1 0c ac bd c2 79 6c c8 bf 2a 06 95 70 88 5e 63 9b 08 5f 95 ca 5a 52 46 45 43 9b 36 ae 1b 20 8e 5a d3 e6 33 27 f1 c3 94 8f ab a8 99 2c d4 58 a9 91 97 0a 3c e5 4a c6 7a c9 17 1f c1 38 67 e4 b6 28 4a 0a 51 53 65 a1 fa c4 5b 32 0b 28 5c ac 2c a5 8e 62 c2 15 47

Code file example:

	PKCS #7 Digital Signature {
30 82 0b c1	ContentInfo header
06 09 2a 86 48 86 f7 0d 01 07 02	ContentType=signedData
a0 82 0b b2	[0] EXPLICIT
30 82 0b ae	SignedData header
02 01 01	Version=1
31 0b 30 09 06 05 2b 0e 03 02 1a 05 00	SET OF DigestAlgorithmIdentifier DigestAlgorithmIdentifier header Algorithm=SHA-1 Parameters=NULL
30 0b 06 09 2a 86 48 86 f7 0d 01 07 01	ContentInfo header contentType=data
a0 82 07 2a	Certificate(s) 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.
30 82 03 89	Manufacturer CVC header
30 82 02 71	tbs Manufacturer CVC header
a0 03 02 01 02	Version=v3(2)
02 05 01 02 03 04 05	Serial Number = 01 02 03 04 05
30 0d 06 09 2a 86 48 86 f7 0d 01 01 05 05 00	Signature Algorithm=SHA-1 with RSA OID Parameters=NULL
30 81 97	Issuer SEQUENCE
31 0b 30 09 06 03 55 04 06 13 02 55 53	 AttributeType=countryName AttributeValue="US"
31 39 30 37 06 03 55 04 0a 13 30 44 61 74 61 20 4f 76 65 72 20 43 61 62 6c 65 20 53 65 72 76 69 63 65 20 49 6e 74 65 72 66 61 63 65 20 53 70 65 63 69 66 69 63 61 74 69 6f 6e 73	 AttributeType=organizationalName AttributeValue= "Data Over Cable Service Interface Specifications"
31 15 30 13 06 03 55 04 0b 13 0c 43 61 62 6c 65 20 4d 6f 64 65 6d 73	 AttributeType= organizationalUnitName AttributeValue= "Cable Modems"

31 36 30 34 06 03 55 04 03 13 2d 44 4f 43 53 49 53 20 43 61 62 6c 65 20 4d 6f 64 65 6d 20 52 6f 6f 74 20 43 65 72 74 69 66 69 63 61 74 65 20 41 75 74 68 6f 72 69 74 79	 AttributeType=commonName AttributeValue= “DOCSIS Cable Modem Root Certificate Authority”
30 1e 17 0d 30 31 30 31 30 31 30 30 30 30 30 30 5a 17 0d 31 31 30 31 30 31 30 30 30 30 30 30 5a	Validity SEQUENCE Not before=2001/01/01:00:00:00Z Not after=2011/01/01:00:00:00Z
30 58	Subject SEQUENCE
31 0b 30 09 06 03 55 04 06 13 02 55 53	 AttributeType=countryName AttributeValue=”US”
31 10 30 0e 06 03 55 04 0a 13 07 56 65 6e 64 6f 72 41	 AttributeType=organizationalName AttributeValue=“VendorA”
31 0f 30 0d 06 03 55 04 0b 13 06 44 4f 43 53 49 53	 AttributeType= organizationalUnitName AttributeValue=“DOCSIS”
31 26 30 24 06 03 55 04 03 13 1d 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 AttributeValue= “Code Verification Certificate”
30 82 01 22	SubjectPublicKeyInfo header
30 0d 06 09 2a 86 48 86 f7 0d 01 01 01 05 00	Public Key SEQUENCE Algorithm=RSA encryption Parameters=NULL
03 82 01 0f 00 30 82 01 0a	Public Key header
02 82 01 01	Public Key modulus header
00 a2 05 d2 10 02 f3 47 b4 dd 2e 2a 9b 05 b3 86 38 a4 91 6d ad 39 5a da 0b 48 55 9b 94 dd 4e 9d 8b 98 c6 18 6d e8 de 1e 81 db ce 8f 3c 6d 02 81 fa 1e 46 f1 93 30 c2 79 bb ba f6 50 b2 db 10 b2 13 c0 bd a5 4d 33 5f 43 0d 9f a0 de 9b 8b 71 b5 ff 1f 48 e1 06 38 71 b8 df 3c 93 c6 f8 eb 90 a6 d2 a4 aa 7b 4e f1 2e 5c 11 07 88 1d 13 f8 60 d8 e2 3b e5 c7 7b 30 39 74 75 94 e3 73 19 92 b6 81 2f b7 db 85 be 6a 8e 54 e9 dd 67 8f a7 a3 4c d6 22 50 a8 0d 69 41 98 d5 a3 af 51 8d 48 44 f4 35 9d 5d 7d a3 d1 62 e4 01 3f 62 05 e1 89 57 5e 5b 90 09 ff 83 7b b6 f2 0d 32 72 47 9e e2 d6 1a 15 c0 97 53 9a a8 9f 24 5c 88 cf bb 98 b4 da 01 73 33 31 ea aa 8d e2 01 9f 20 3e e1 67 dd 36 2e 3c 83 28 5d 28 bb 6a d9 30 94 37 e6 d8 86 42 32 c7 9b 36 68 65 3b 3f 40 f3 87 1a 31 92 90 93 a3 21 b1	Public Key modulus
02 03 01 00 01	Public Key exponent

a3 1a 30 18 30 16 06 03 55 1d 25 01 01 ff 04 0c 30 0a 06 08 2b 06 01 05 05 07 03 03	[3] Extensions OPTIONAL Extensions header SEQUENCE Extension header SEQUENCE Extended Key Usage Critical=TRUE keyPurposeID=codeSigning
30 0d 06 09 2a 86 48 86 f7 0d 01 01 05 05 00	Signature SEQUENCE Algorithm=SHA-1 with RSA OID Parameters=NULL
03 82 01 01 00	Signature value header
74 05 b9 35 50 3d 41 a1 da 36 6f 8f 31 bf aa b9 b4 2b 93 19 1b f1 57 3d 1e 79 b2 22 0b c1 c8 20 80 5a 0a 8d b1 12 46 7f 8d f4 ee ca 51 87 ba b0 7b 67 ba db 57 d6 bd d7 3a c3 6c b8 39 dc 85 25 d3 9a 07 f4 98 ff ce e7 de a9 f5 06 89 dc e0 af 39 e4 05 34 2f ac ae 49 5e 90 a6 f4 2b 4c 4c 9d 2c a1 9e 75 d8 85 5f c5 c4 d8 b1 d1 ec e7 cc 77 d0 75 73 08 c4 88 48 25 2c 60 69 37 11 1f 3e 6f e5 27 2a 8e c3 9e 9a c8 18 23 32 7d d9 78 87 7c 5a 7d b8 36 77 32 03 32 20 5c 7b 49 eb ec 01 ea b4 ba 5a f2 1f f7 c7 d8 3c 0e aa 49 10 15 14 85 db d3 18 61 e5 94 29 5a 48 13 82 8b 5a e8 95 7c 09 ce 11 e1 49 78 28 52 4f bc 33 63 4a aa c4 01 0a 59 f6 42 af 89 c0 cd e0 4b 94 c2 48 8c c3 25 d7 44 2e af 87 7b 60 17 f1 f7 4b 54 56 a3 33 2d 69 e4 af de ae 01 e6 ab 2b a7 61 38 15 4c 5d 29	Signature value
30 82 03 99	Co-signer CVC header
30 82 02 81	tbs Co-signer CVC header
a0 03 02 01 02	Version=v3(2)
02 14 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14	Serial Number = 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 11 12 13 14
30 0d 06 09 2a 86 48 86 f7 0d 01 01 05 05 00	Signature Algorithm=SHA-1 with RSA OID Parameters=NULL
30 81 97	Issuer SEQUENCE
31 0b 30 09 06 03 55 04 06 13 02 55 53	 AttributeType=countryName AttributeValue="US"
31 39 30 37 06 03 55 04 0a 13 30 44 61 74 61 20 4f 76 65 72 20 43 61 62 6c 65 20 53 65 72 76 69 63 65 20 49 6e 74 65 72 66 61 63 65 20 53 70 65 63 69 66 69 63 61 74 69 6f 6e 73	 AttributeType=organizationalName AttributeValue= "Data Over Cable Service Interface Specifications"
31 15 30 13 06 03 55 04 0b 13 0c 43 61 62 6c 65 20 4d 6f 64 65 6d 73	 AttributeType= organizationalUnitName AttributeValue= "Cable Modems"

31 36 30 34 06 03 55 04 03 13 2d 44 4f 43 53 49 53 20 43 61 62 6c 65 20 4d 6f 64 65 6d 20 52 6f 6f 74 20 43 65 72 74 69 66 69 63 61 74 65 20 41 75 74 68 6f 72 69 74 79	 AttributeType=commonName AttributeValue= "DOCSIS Cable Modem Root Certificate Authority"
30 1e 17 0d 30 31 30 31 30 31 30 30 30 30 30 30 5a 17 0d 31 30 30 31 30 31 30 30 30 30 30 30 5a	Validity SEQUENCE Not before=2001/01/01:00:00:00Z Not after=2010/01/01:00:00:00Z
30 59	Subject SEQUENCE
31 0b 30 09 06 03 55 04 06 13 02 55 53	 AttributeType=countryName AttributeValue="US"
31 11 30 0f 06 03 55 04 0a 13 08 41 42 43 44 45 46 30 31	 AttributeType=organizationalName AttributeValue= "ABCDEF01"
31 0f 30 0d 06 03 55 04 0b 13 06 44 4f 43 53 49 53	 AttributeType= organizationalUnitName AttributeValue= "DOCSIS"
31 26 30 24 06 03 55 04 03 13 1d 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 AttributeValue= "Code Verification Certificate"
30 82 01 22	SubjectPublicKeyInfo header
30 0d 06 09 2a 86 48 86 f7 0d 01 01 01 05 00	Public Key SEQUENCE Algorithm=RSA encryption Parameters=NULL
03 82 01 0f 00 30 82 01 0a	Public Key header
02 82 01 01	Public Key modulus header
00 ba d2 ed 99 85 b8 c5 6f d8 c2 20 40 20 19 ff ef ec 79 16 4f 21 93 47 dd 03 b2 41 d0 2e 85 a8 46 8e 5d 83 e7 10 43 a8 bd c2 00 cd a4 9b a7 8b 46 91 ea 64 2e 56 79 b2 84 7f ff 0e 91 9a 12 fc 99 af 92 f3 70 48 a8 50 ab 0d b9 41 65 48 f5 c8 78 23 0d 6b 57 60 ec 64 4d 10 7f 05 49 1e ec 8c 86 fd 5e 91 f5 34 3e dc 21 3e 99 6c e8 68 c3 9a f6 38 30 c2 ac 0e b4 5a 50 df d4 8c 5d 08 8d 8f 97 91 e3 6f 7b 6d 1e 16 b3 38 29 8b ef 4e b2 ea 02 f0 4d 28 f6 7a 85 8d 8c 04 ed 73 d6 f7 fd 01 74 a8 6e 82 d9 e7 b2 15 9d 9e c2 83 cb bf 77 e5 69 18 d5 4a e5 91 17 08 40 61 9d a9 77 37 6b 0f e2 3a 7d 35 d1 1f be 3a ea ed e6 fd de e0 f2 7f 79 3b a1 f0 26 8e 69 97 44 b9 e3 4d 24 8f 1e 26 1d 0c fa bc cd 2b e4 e6 f0 54 55 94 e5 6b 26 dc 30 5f fc 3e 1e cd 2d 28 36 58 bc f3 f8 f0 f5 4a e7	Public Key modulus

02 03 01 00 01	Public Key exponent
a3 1a 30 18 30 16 06 03 55 1d 25 01 01 ff 04 0c 30 0a 06 08 2b 06 01 05 05 07 03 03	[3] Extensions OPTIONAL Extensions header SEQUENCE Extension header SEQUENCE Extended Key Usage Critical=TRUE keyPurposeID=codeSigning
30 0d 06 09 2a 86 48 86 f7 0d 01 01 05 05 00	Signature SEQUENCE Algorithm=SHA-1 with RSA OID Parameters=NULL
03 82 01 01 00	Signature value header
ae 2e 2e 42 23 fc ee d9 91 8f c6 a4 71 a4 8c d8 7f 17 18 8a 0b 85 8c 5d 08 51 0c 4d 44 4f f9 4d 94 e9 e9 23 5e a2 29 d7 95 1d 16 30 9d ac 44 a5 d8 7d 19 5f c0 87 b3 06 56 93 0c 02 40 8d cf 6d fb 93 39 01 1a 17 1a 49 ed 25 d2 5b 4f f5 a4 24 38 a2 5a 8b ea 4b de 82 6a b4 e2 8f e3 e8 24 71 6e 90 3a 0b 29 5c a0 44 fe ed af 6b 80 4f eb 13 a7 75 8f be 3c f2 e2 ef e2 44 f2 40 45 76 70 d6 40 b4 51 7f fe 42 20 65 90 04 1a f6 92 07 a1 1c 72 4f 04 0d 3a 1c 02 01 94 fc 0e aa cd 91 f7 4f f0 03 e8 be 12 e9 6a 17 37 0e ed e0 b5 1d 5f e6 af 6a bb 0b e9 6b c3 11 6f 27 ac 60 70 0d cc 21 98 b3 dd a9 3c be ca 55 02 e8 b2 f9 5c 18 c1 5a db d1 d9 e8 9b 7f 36 0c 86 5e 93 49 a1 e8 f5 42 4b c5 82 f8 da 6a f5 05 78 69 8d a8 ec 50 81 e0 5b 0a be 94 ae 7c d1 67 43 28 5c d3 43 42 c7 5a	Signature value
31 82 04 5f	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, the this size is the total size of the Manufacturer SignerInfo plus the total size of the Co-signer SignerInfo.
30 82 02 24	Manufacturer SignerInfo header
02 01 01	Version=1
30 81 a1	IssuerAndSerialNumber header
30 81 97	Issuer Name header
31 0b 30 09 06 03 55 04 06 13 02 55 53	 AttributeType=countryName AttributeValue="US"
31 39 30 37 06 03 55 04 0a 13 30 44 61 74 61 20 4f 76 65 72 20 43 61 62 6c 65 20 53 65 72 76 69 63 65 20 49 6e 74 65 72 66 61 63 65 20 53 70 65 63 69 66 69 63 61 74 69 6f 6e 73	 AttributeType=organizationalName AttributeValue= "Data Over Cable Service Interface Specifications"

31 15 30 13 06 03 55 04 0b 13 0c 43 61 62 6c 65 20 4d 6f 64 65 6d 73	 AttributeType= organizationalUnitName AttributeValue= “Cable Modems”
31 36 30 34 06 03 55 04 03 13 2d 44 4f 43 53 49 53 20 43 61 62 6c 65 20 4d 6f 64 65 6d 20 52 6f 6f 74 20 43 65 72 74 69 66 69 63 61 74 65 20 41 75 74 68 6f 72 69 74 79	 AttributeType=commonName AttributeValue= “DOCSIS Cable Modem Root Certificate Authority”
02 05 01 02 03 04 05	CertificateSerialNumber = 01 02 03 04 05
30 09 06 05 2b 0e 03 02 1a 05 00	DigestAlgorithmIdentifier header Algorithm=SHA-1 Parameters=NULL
a0 5d	[0] AuthenticateAttributes
30 18 06 09 2a 86 48 86 f7 0d 01 09 03 31 0b 06 09 2a 86 48 86 f7 0d 01 07 01	ContentType header AttributeType=contentType SET OF AttributeValue=Data
30 1c 06 09 2a 86 48 86 f7 0d 01 09 05 31 0f 17 0d 30 32 30 31 30 31 30 30 30 30 30 30 5a	SigningTime header AttributeType=signingType SET OF attributeValue= 2002/01/01/00:00:00Z
30 23 06 09 2a 86 48 86 f7 0d 01 09 04 31 16 04 14 a9 1e 19 e0 b7 5a 2d 80 bb 78 eb 9b d7 70 97 6d 01 30 6c 8a	MessageDigest header AttributeType=messageDigest SET OF attributeValue=OCTET STRING Message Digest
30 0d 06 09 2a 86 48 86 f7 0d 01 01 01 05 00	DigestEncryptionAlgorithm header Algorithm=RSA Parameters=NULL
04 82 01 00 2b ee fd 17 67 4a 5a 74 26 b6 1f 74 fb 01 f4 c4 77 d6 b2 db 42 c7 0c e2 e5 25 28 16 cb c5 43 e5 f9 fe 92 76 05 91 b0 cc 00 19 7c 8e c2 17 8c 95 96 15 1c 50 38 53 99 9d 1a 41 18 fc 3c 39 c0 e6 ae 26 e1 09 07 a2 75 a5 2e f9 3e ee 87 51 11 6c a7 22 fb c8 8a 52 6f fc 76 56 b4 c3 bb bf 17 56 c7 5c 8b a7 20 c4 4b 75 d8 72 69 71 e2 60 06 3d 43 52 cc d7 20 20 19 86 21 da 58 90 f0 15 ae 30 26 aa 05 4d 8a 3f af 31 f3 26 66 53 4a e2 0b 3e 03 a1 9f 7a 0d 87 09 fe ac 3c 07 45 07 f3 30 f2 28 02 27 f6 fe 46 fb de a0 01 77 50 39 53 97 ad 6b 86 33 76 0e 5e d6 ba 66 c3 0e 0d 56 c8 c8 5a dd d2 2f 4d e8 b7 44 7e 53 b4 15 f8 b0 61 8f 73 3c 66 75 ff e6 d4 98 f1 8d 43 76 02 35 bc 16 1c 16 88 ed 31 87 fd 7f e6 50 6b 9a 56 48 6d 8c f8 4d 02 5c 29 f5 6e 1e 01 cb 3d 70 e3 e1 2f 44 a7	OCTET STRING EncryptedDigest

30 82 02 33	Co-signer SignerInfo header
02 01 01	Version=1
30 81 b0	IssuerAndSerialNumber header
30 81 97	Issuer Name header
31 0b 30 09 06 03 55 04 06 13 02 55 53	 AttributeType=countryName AttributeValue="US"
31 39 30 37 06 03 55 04 0a 13 30 44 61 74 61 20 4f 76 65 72 20 43 61 62 6c 65 20 53 65 72 76 69 63 65 20 49 6e 74 65 72 66 61 63 65 20 53 70 65 63 69 66 69 63 61 74 69 6f 6e 73	 AttributeType=organizationalName AttributeValue= "Data Over Cable Service Interface Specifications"
31 15 30 13 06 03 55 04 0b 13 0c 43 61 62 6c 65 20 4d 6f 64 65 6d 73	 AttributeType= organizationalUnitName AttributeValue= "Cable Modems"
31 36 30 34 06 03 55 04 03 13 2d 44 4f 43 53 49 53 20 43 61 62 6c 65 20 4d 6f 64 65 6d 20 52 6f 6f 74 20 43 65 72 74 69 66 69 63 61 74 65 20 41 75 74 68 6f 72 69 74 79	 AttributeType=commonName AttributeValue= "DOCSIS Cable Modem Root Certificate Authority"
02 14 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14	CertificateSerialNumber = 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 11 12 13 14
30 09 06 05 2b 0e 03 02 1a 05 00	DigestAlgorithmIdentifier header Algorithm=SHA-1 Parameters=NULL
a0 5d	[0] AuthenticateAttributes
30 18 06 09 2a 86 48 86 f7 0d 01 09 03 31 0b 06 09 2a 86 48 86 f7 0d 01 07 01	ContentType header AttributeType=contentType SET OF AttributeValue=Data
30 1c 06 09 2a 86 48 86 f7 0d 01 09 05 31 0f 17 0d 30 32 30 31 30 31 30 30 30 30 30 30 5a	SigningTime header AttributeType=signingType SET OF attributeValue= 2002/01/01/00:00:00Z
30 23 06 09 2a 86 48 86 f7 0d 01 09 04 31 16 04 14 a9 1e 19 e0 b7 5a 2d 80 bb 78 eb 9b d7 70 97 6d 01 30 6c 8a	MessageDigest header AttributeType=messageDigest SET OF attributeValue=OCTET STRING Message Digest

30 0d 06 09 2a 86 48 86 f7 0d 01 01 01 05 00	DigestEncryptionAlgorithm header Algorithm=RSA Parameters=NULL
04 82 01 00 b9 0b ee f5 d3 bb c2 30 fb 8e 64 99 b6 27 f1 44 2e 33 30 27 c3 25 5d 2a c0 79 68 5a b8 50 99 a6 b5 27 ef 82 7b ad a7 e6 2b 7f f5 39 32 35 37 12 f3 db 49 06 1a 9c 25 20 68 ac 3b 7f 6c 31 92 3a 5e ec 15 a5 50 e4 db 3d 41 90 a8 f7 2c 0a 42 da 86 1e b3 cd f1 67 c1 27 8d 62 0c 19 08 08 38 19 aa ec 67 20 a2 b4 73 46 50 4f 5c 6e 0c 5d cd ed df b5 20 85 74 2c 34 53 9a 7f 85 2f f4 7d f2 55 c1 f5 3a 09 97 54 6e f2 6c 26 0e 77 98 d6 68 cd 96 7d 19 0f 1f 8f c3 ea 70 a9 46 69 3d ed 23 b5 e1 39 a5 68 1c 0a da ac 95 16 93 8e e1 13 ed dc 52 69 fa d4 2f 7d c6 31 05 46 0b 19 48 28 18 05 85 34 4b e5 d3 18 4a 2f 53 b6 bb 40 95 0d 88 ff d7 94 d9 ec 15 ea 70 b6 ab 91 a6 4c 69 87 b1 ac e0 c5 d0 71 7d a3 e6 7e 59 08 eb 11 c6 c6 3f e1 29 01 c5 fc 34 b6 b9 f0 aa 64 86 78 3d f2 aa 9d	OCTET STRING EncryptedDigest
	} end of PKCS #7 Digital Signature
	SignedContent {
1c 0a ce	Mandatory Download Parameters TLV format (Type 28) {
04 01 0e	Optional DOCSIS Root CA Public Key TLV format(Type 4) {
30 82 01 0a	Public Key header
02 82 01 01	Public Key modulus header
00 c3 10 35 70 ad 35 a7 60 a8 a5 29 c7 11 2f 2b fc 14 32 39 b3 35 0d 64 2f ae ce fb 56 53 9f b6 ea b1 d2 22 4f 5c 12 7c ea 0a b2 7d 0a 0e fb 93 9d e5 cd 3b 18 5c 61 37 50 9a ec 60 3d f0 07 41 03 db ec 34 10 42 7c fd 7d 6d 08 62 1c 43 b7 b0 72 a1 c8 c2 cb 60 d5 ac 99 61 05 3d 3d 2f 70 ea c1 77 77 dd 8e c8 b6 0f b7 d1 b4 c5 a3 da 02 bb 9c 5f 38 93 b9 a0 b5 3a 95 34 66 79 24 16 ea 19 59 d5 d7 4f 06 b3 39 32 a5 ef d8 ff c9 92 f9 82 4c 93 ad 52 45 de 69 92 ca 86 f5 b6 8b 98 7e c1 32 24 57 5a 9d 45 1c 7c e5 b0 dd 22 b7 25 cb 24 45 ef 34 14 a5 3f a1 e4 1e c3 3d 9e d0 e4 48 4b aa 62 69 da 79 fe 74 c7 72 eb 3e 91 d6 77 35 e0 4b 55 a4 03 27 ef c0 f1 9b 68 92 47 f9 af ae 2e 51 26 76 20 aa 98 33 d2 36 4c 46 b7 8c 21 ea b1 2d 18 14 d4 fc 72 c3 be 22 2f 32 13 9d dc a3 4e 0f	Public Key modulus
02 03 01 00 01	Public Key exponent
	} End of DOCSIS Root CA Public Key
11 03 01	Optional Manufacturer CA Certificate TLV format (Type 17) {
30 82 02 fd	Manufacturer CA Certificate header
30 82 01 e5	tbs Manufacturer CA Certificate header
a0 03 02 01 02	Version=v3(2)
02 08 11 22 33 44 55 66 77 88	Serial Number = 11 22 33 44 55 66 77 88
30 0d 06 09 2a 86 48 86 f7 0d 01 01 05 05 00	Signature Algorithm=SHA-1 with RSA OID Parameters=NULL
30 81 97	Issuer SEQUENCE

31 0b 30 09 06 03 55 04 06 13 02 55 53	AttributeType=countryName AttributeValue="US"
31 39 30 37 06 03 55 04 0a 13 30 44 61 74 61 20 4f 76 65 72 20 43 61 62 6c 65 20 53 65 72 76 69 63 65 20 49 6e 74 65 72 66 61 63 65 20 53 70 65 63 69 66 69 63 61 74 69 6f 6e 73	AttributeType=organizationalName AttributeValue= "Data Over Cable Service Interface Specifications"
31 15 30 13 06 03 55 04 0b 13 0c 43 61 62 6c 65 20 4d 6f 64 65 6d 73	AttributeType= organizationalUnitName AttributeValue= "Cable Modems"
31 36 30 34 06 03 55 04 03 13 2d 44 4f 43 53 49 53 20 43 61 62 6c 65 20 4d 6f 64 65 6d 20 52 6f 6f 74 20 43 65 72 74 69 66 69 63 61 74 65 20 41 75 74 68 6f 72 69 74 79	AttributeType=commonName AttributeValue= "DOCSIS Cable Modem Root Certificate Authority"
30 1e 17 0d 30 31 30 31 30 31 30 30 30 30 30 30 5a 17 0d 32 30 30 31 30 31 30 30 30 30 30 30 5a	Validity SEQUENCE Not before=2001/01/01:00:00:00Z Not after=2020/01/01:00:00:00Z
30 69	Subject SEQUENCE
31 0b 30 09 06 03 55 04 06 13 02 55 53	AttributeType=countryName AttributeValue="US"
31 10 30 0e 06 03 55 04 0a 13 07 56 65 6e 64 6f 72 41	AttributeType=organizationalName AttributeValue="VendorA"
31 0f 30 0d 06 03 55 04 0b 13 06 44 4f 43 53 49 53	AttributeType= organizationalUnitName AttributeValue="DOCSIS"
31 37 30 35 06 03 55 04 03 13 2e 56 65 6e 64 6f 72 41 20 43 61 62 6c 65 20 4d 6f 64 65 6d 20 52 6f 6f 74 20 43 65 72 74 69 66 69 63 61 74 65 20 41 75 74 68 6f 72 69 74 79	AttributeType=commonName AttributeValue= "VendorA Cable Modem Root Certificate Authority"
30 81 9f	SubjectPublicKeyInfo header
30 0d 06 09 2a 86 48 86 f7 0d 01 01 01 05 00	Public Key SEQUENCE Algorithm=RSA encryption Parameters=NULL
03 81 8d 00 30 81 89	Public Key header
02 81 81	Public Key modulus header

00 9e 2d b0 50 74 22 2a 6c 80 c1 3e 27 f1 bf 94 c2 e0 3e 03 2d 71 35 6d 0e 9c 64 25 08 01 7c c6 7b ee 21 91 6c 16 0a 91 94 78 70 e3 04 7e a1 c4 bf 36 f9 26 97 a2 59 85 a9 25 47 8b 20 6b bf fd f0 f0 83 fb 7f 83 02 a1 55 48 e6 1b 6b f2 8e 2a b6 a7 7a d4 ee 84 24 47 c0 8c 8e 8d e8 d4 5c da 58 2c a1 f7 3d 15 88 a3 6a 5c ea 45 be 8e cf fd cd 50 d7 d4 55 2e 2a 29 47 dc 33 84 fb e4 b2 a8 a3	Public Key modulus
02 03 01 00 01	Public Key exponent
30 0d 06 09 2a 86 48 86 f7 0d 01 01 05 05 00	Signature SEQUENCE Algorithm=SHA-1 with RSA OID Parameters=NULL
03 82 01 01 00	Signature value header
b2 2e b9 e1 53 83 b9 b7 a4 c3 d4 4e 39 24 63 db eb 1e b1 c0 ce be 56 92 d6 2e d6 8a 3b 80 f7 2b 1d e7 e3 6d 17 51 20 06 53 d3 ec 5a b1 98 76 d2 6c c3 cc bf 0b 7f 77 02 35 ce 4f d9 71 a2 c3 1b c2 70 7c 27 bd 3c b4 98 74 c5 a0 d1 de 4e 86 49 95 5f 38 36 9a 15 30 31 64 89 a1 ee e3 70 b7 60 fd 79 32 52 83 66 63 e5 a3 0e e3 8c 39 f7 6d 81 35 bc 64 b1 f4 fe 86 7a 9e dc c0 86 b4 f9 5d de b2 60 d7 01 fb 88 5b 44 2f c9 f3 e4 01 a8 69 70 31 c1 d5 a3 6a 35 bd e4 00 31 15 15 55 6e e4 cd 19 13 4f 60 ba ae 4a b4 63 c7 95 da 88 3f 0a 2b d1 89 aa 18 05 d5 94 91 75 da a3 1d a7 1c 75 96 6a e2 5a 2a dc 3b 3a b9 69 fe 20 73 2b c1 7f 4b b6 7f bc 0d 4f da 33 ba 3d 0f bf d2 6d cb d2 01 ba 46 ea f3 2a a0 93 a1 b0 57 1b d9 cd b0 22 97 2c f7 1e dc 22 30 2b ae 72 68 dc 11 ac 13 80 f4	Signature value
	} End of Manufacturer CA Certificate
33 03 4a	Optional CVC Root CA Certificate TLV format (Type 51) {
30 82 03 46	CVC Root CA Certificate header
30 82 02 2e	tbs CVC Root CA Certificate header
a0 03 02 01 02	Version=v3(2)
02 08 01 02 03 04 05 06 07 08	Serial Number = 01 02 03 04 05 06 07 08
30 0d 06 09 2a 86 48 86 f7 0d 01 01 05 05 00	Signature Algorithm=SHA-1 with RSA OID Parameters=NULL
30 41	Issuer SEQUENCE
31 0b 30 09 06 03 55 04 06 13 02 55 53	 AttributeType=countryName AttributeValue="US"
31 12 30 10 06 03 55 04 0a 13 09 43 61 62 6c 65 4c 61 62 73	 AttributeType=organizationalName AttributeValue= "CableLabs"
31 1e 30 1c 06 03 55 04 03 13 15 43 61 62 6c 65 4c 61 62 73 20 43 56 43 20 52 6f 6f 74 20 43 41	 AttributeType= organizationalUnitName AttributeValue= "'CableLabs CVC Root CA"
30 1e 17 0d 30 32 30 36 30 32 32 30 31 36 31 30 5a 17 0d 33 32 30 36 30 32 30 38 31 36 31 30 5a	Validity SEQUENCE Not before=2002/06/02:20:16:10Z Not after=2032/06/02:08:16:10Z

30 41	Subject SEQUENCE
31 0b 30 09 06 03 55 04 06 13 02 55 53	 AttributeType=countryName AttributeValue=”US”
31 12 30 10 06 03 55 04 0a 13 09 43 61 62 6c 65 4c 61 62 73	 AttributeType=organizationalName AttributeValue= “CableLabs”
31 1e 30 1c 06 03 55 04 03 13 15 43 61 62 6c 65 4c 61 62 73 20 43 56 43 20 52 6f 6f 74 20 43 41	 AttributeType= organizationalUnitName AttributeValue= “'CableLabs CVC Root CA”
30 82 01 22	SubjectPublicKeyInfo header
30 0d 06 09 2a 86 48 86 f7 0d 01 01 01 05 00	Public Key SEQUENCE Algorithm=RSA encryption Parameters=NULL
03 82 01 0f 00 30 82 01 0a	Public Key header
02 82 01 01	Public Key modulus header
00 c7 52 7e e7 f4 aa 0b 79 e5 8c 2b 1b fb d2 94 77 30 23 03 2a 14 21 7e b2 6d 3f 16 c9 0d c7 f2 92 25 ba 38 90 0f c9 c8 7e 94 c2 b8 e7 e1 79 5e 41 c9 c4 23 8f 76 0f f1 d5 47 f6 ea bd 6f 9d 21 0b 9d 85 a6 ee 71 04 26 e1 16 34 42 89 ab e3 fb aa 28 c0 b2 0c 6b 58 d4 08 c5 ff b0 57 20 c8 b2 c7 be 89 2f 1e 8c a0 02 bb fc 58 b1 cc eb 05 97 e4 b1 e4 b9 7f bc fc 86 04 9f 4d 29 50 d5 eb 1c 74 ae b4 ad 4c 2b 3a ac 0b f2 a2 34 2f cc b0 88 9f d6 f5 08 95 b7 4c bb eb 5b ae 3d 4f 01 74 bf 93 89 06 b1 6d c8 6e 9d 37 16 8e 3d 06 1c a0 ac 32 c9 9f 6c 3f 9e b0 9e 65 9e 83 01 25 10 e3 78 92 59 89 ff 6f 7b e3 b0 bc bb ab 4d 86 cc a0 68 08 df b5 41 b0 a7 0e 61 d3 cf ae 3c 7d 50 e3 f2 34 f8 aa ac 19 d8 93 4f ba d1 f1 5e aa 73 11 ea cd e0 2a 1a d1 fc 07 7b c3 11 ce b0 d0 29 6e 5c 23	Public Key modulus
02 03 01 00 01	Public Key exponent
a3 42	[3] Extensions
30 40	Extensions header SEQUENCE
30 1d 06 03 55 1d 0e 04 16 04 14 08 19 b7 ba 13 2a 4d 2f d8 fe ce 07 b3 ab f9 5b 35 92 4b 11	Extension header SEQUENCE subjectKeyIdentifier OCTET STRING encapsulated OCTET STRING Subject Key Identifier
30 0e 06 03 55 1d 0f 01 01 ff 04 04 03 02 01 06	Extension header SEQUENCE Extended Key Usage Critical=TRUE Certificate Signing, Off-line CRL Signing, CRL Signing
30 0f 06 03 55 1d 13 01 01 ff 04 05 30 03 01 01 ff	Extension header SEQUENCE basicConstraints Critical=TRUE Subject Type=CA, Path Length Constraint=None

30 0d 06 09 2a 86 48 86 f7 0d 01 01 05 05 00	Signature SEQUENCE Algorithm=SHA-1 with RSA OID Parameters=NULL
03 82 01 01 00	Signature value header
0e 9a 18 13 b8 09 af 84 0e 57 a8 5f 4b 50 83 4d af 24 c4 e1 99 c1 b7 cd 04 4c 0a 4a 86 28 92 99 8d dd 7e 9f 2e 5f 7b 9d 3f c6 9f ff f7 75 0a f6 4b 2b c1 45 de 9f 9b be 15 c7 ba b2 3a 23 22 3b ec 9e 59 8f 19 cd 66 e2 a6 d5 e9 0a 5e 60 41 3e 58 e0 63 f3 89 0e 7a 33 2b f3 c9 41 77 16 b1 cb 1e a1 5e a0 b6 d3 76 7d e9 b4 18 d0 5d 1d 03 50 17 a0 cd 5e 28 93 5d ad 39 c9 fb f1 15 43 9a 77 90 3b 94 a3 43 a2 f3 47 94 7c f5 59 66 ae 90 65 f1 c1 6c ae 5f cc 26 2a 84 a3 24 24 c8 42 a5 2b 88 94 77 43 bf af 67 b4 68 e2 4d 9b 05 7d be b9 86 81 88 27 de c6 f2 f6 0a 73 e6 f9 41 22 45 02 bd 6f 95 40 d5 5a fc e3 5b 32 ca 26 63 52 51 25 b3 24 8d f5 3f 35 ca 99 32 d2 bd 3f 00 24 27 f1 93 f0 6f 0e 7a ff fb e6 02 4e b3 a2 90 69 53 c3 16 20 18 76 7f c4 06 a4 e4 8b 35 e7 d9 13 49 65	Signature value
	} End of CVC Root CA Certificate
34 03 69	Optional CVC CA Certificate TLV format (Type 52) {
30 82 03 65	CVC CA Certificate header
30 82 02 4d	tbs CVC CA Certificate header
a0 03 02 01 02	Version=v3(2)
02 08 11 22 33 44 55 66 77 88	Serial Number = 11 22 33 44 55 66 77 88
30 0d 06 09 2a 86 48 86 f7 0d 01 01 05 05 00	Signature Algorithm=SHA-1 with RSA OID Parameters=NULL
30 41	Issuer SEQUENCE
31 0b 30 09 06 03 55 04 06 13 02 55 53	 AttributeType=countryName AttributeValue="US"
31 12 30 10 06 03 55 04 0a 13 09 43 61 62 6c 65 4c 61 62 73	 AttributeType=organizationalName AttributeValue= "CableLabs"
31 1e 30 1c 06 03 55 04 03 13 15 43 61 62 6c 65 4c 61 62 73 20 43 56 43 20 52 6f 6f 74 20 43 41	 AttributeType= organizationalUnitName AttributeValue= "'CableLabs CVC Root CA"
30 1e 17 0d 30 32 30 36 30 32 32 30 35 31 35 35 5a 17 0d 32 32 30 36 30 32 32 30 35 31 35 35 5a	Validity SEQUENCE Not before=2002/06/02:20:51:55Z Not after=2022/06/02:20:51:55Z
30 3c	Subject SEQUENCE
31 0b 30 09 06 03 55 04 06 13 02 55 53	 AttributeType=countryName AttributeValue="US"
31 12 30 10 06 03 55 04 0a 13 09 43 61 62 6c 65 4c 61 62 73	 AttributeType=organizationalName AttributeValue= "CableLabs"

31 19 30 17 06 03 55 04 03 13 10 43 61 62 6c 65 4c 61 62 73 20 43 56 43 20 43 41	 AttributeType= organizationalUnitName AttributeValue= "'CableLabs CVC CA"
30 82 01 22	SubjectPublicKeyInfo header
30 0d 06 09 2a 86 48 86 f7 0d 01 01 01 05 00	Public Key SEQUENCE Algorithm=RSA encryption Parameters=NULL
03 82 01 0f 00 30 82 01 0a	Public Key header
02 82 01 01	Public Key modulus header
00 ce 44 c6 b2 91 48 be 93 b6 d0 43 4c 89 ec 87 1e e3 de 8c 3a 8e 02 e2 df 4f 5a b6 8d 5d 36 96 4e 72 62 33 d2 bf 54 21 92 a8 3c ef 7e d5 75 99 d2 92 85 0c b7 e3 c9 a8 a9 57 30 19 00 68 09 44 69 db 28 02 eb 63 b2 79 b5 ea 3e 9d d9 8d 4d 2f c3 e9 98 03 34 8b c5 5c 00 27 ba 8c 04 17 56 39 63 3c 82 d8 ca 55 8c 2c cb 9f f2 97 46 c0 ec b7 f8 85 d8 ba d9 60 8b 02 a0 8a b1 ff a7 e7 5a e2 d9 7b 5a 6d 64 8a 4e 90 48 b1 ba 86 26 de 9b 0d 56 9a d2 58 a0 83 02 a1 c4 10 e0 2d e5 2d 01 1c cf 6b 6e 01 3a 64 b8 b2 14 c8 38 d4 14 fe d7 3c d3 6f 37 89 a8 55 fd 4c 02 d4 28 1e bb b3 87 93 da a8 7f 50 5a 27 96 77 1b 54 68 2d 44 00 fa 46 ce 4f 31 69 c5 aa d6 f8 58 5f a9 c8 c6 a4 fe a9 e6 28 9a db c6 d4 57 4a d5 32 8d b0 b5 48 a9 e2 50 0e 95 77 9f 33 87 33 92 2e a6 8d 58 3b 7b 57 dd	Public Key modulus
02 03 01 00 01	Public Key exponent
a3 66	[3] Extensions
30 64	Extensions header SEQUENCE
30 1d 06 03 55 1d 0e 04 16 04 14 2b ea f9 67 5a 78 ae a7 1d bd e9 bb 5a de 1f 7c 11 04 24 c3	Extension header SEQUENCE subjectKeyIdentifier OCTET STRING encapsulated OCTET STRING Subject Key Identifier
30 1f 06 03 55 1d 23 04 18 30 16 80 14 08 19 b7 ba 13 2a 4d 2f d8 fe ce 07 b3 ab f9 5b 35 92 4b 11	Extension header SEQUENCE authorityKeyIdentifier OCTET STRING encapsulated Authority Key Identifier
30 0e 06 03 55 1d 0f 01 01 ff 04 04 03 02 01 06	Extension header SEQUENCE Extended Key Usage Critical=TRUE Certificate Signing, Off-line CRL Signing, CRL Signing
30 12 06 03 55 1d 13 01 01 ff 04 08 30 06 01 01 ff 02 01 00	Extension header SEQUENCE basicConstraints Critical=TRUE Subject Type=CA, Path Length Constraint=0

30 0d 06 09 2a 86 48 86 f7 0d 01 01 05 05 00	Signature SEQUENCE Algorithm=SHA-1 with RSA OID Parameters=NULL
03 82 01 01 00	Signature value header
3e ae f0 46 69 3e 78 94 d0 b3 ab 2f 28 e0 54 1d 14 d2 07 74 eb b4 f5 05 fa 51 f9 87 41 9f cd 95 57 4c ba ca 1d e8 3f de 85 09 ee 88 d3 a2 97 7d ca d9 04 32 7a 70 1b d2 d7 7b e0 8a 80 73 1f 47 d2 f7 d5 66 4a 8e 91 95 7f 41 af 62 e6 6d d2 4d 8a e7 68 0a 3f a1 4e 78 f8 43 07 8d a1 56 f3 3a 14 f5 c6 1c 7b 5f 3b 11 64 96 a6 87 55 a8 ff fc 41 40 6a 56 9f 5d 34 70 49 5a bc 2b 27 95 7c 9d 6f bb e9 5e 7d 4c 05 a0 fe a1 04 97 aa aa bd eb 84 d8 47 da 2f a9 d0 84 59 71 d0 ba 19 c4 4a 65 b9 ac 32 f5 7e 6c 90 f5 f3 ec 06 c1 df 4f ba 06 4f f7 18 1a 71 95 f7 e5 34 ca 05 69 5e 1f 3d 84 d1 f2 68 a5 7f 5b b6 9f 17 02 e0 1f de 7e 53 3f 90 bd 5e 85 37 d4 2f 0c 02 06 74 38 56 fd d2 ba a6 18 3b d2 da 8d 05 10 be 29 9a a7 e3 39 03 dc cd 97 f2 89 59 b3 03 bc 8b b8 35 bb 69 e9 a5 dd	Signature value
	} End of CVC CA Certificate
	} End of Download Parameters

48 65 6c 6c 6f 20 57 6f 72 6c 64 0a	CodeImage() {
	"Hello World"
	}
	Notes: Vendor-specific formatting

	} End of SignedContent

[37]

[37] Table revised per EC SECv3.0-N-06.0322-1 by KN on 11/17/06.

Appendix II Example of Multilinear Modular Hash (MMH) Algorithm Implementation.

This appendix gives an example implementation of the MMH MAC algorithm. There may be other implementations that have advantages over this example in particular operating environments. This example is for informational purposes only and is meant to clarify the specification.

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 must 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 must be even.
The length of the key must be at least (msgLen + 2) bytes.
The length of the pad is four bytes. The pad must 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 must be even.
The length of the key must be at least (msgLen + 6) bytes.
The length of the pad is eight bytes. The pad must 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 Certificate Authority & Provisioning Guidelines

III.1 Certificate Format and Extensions

This section describes the certificate format and extensions used by certificate authority (CA) entities supporting this specification and summarizes the fields of an [X.509], version 3 certificate.

Table III–1 - X.509 Basic Certificate Fields

X.509 v3 Field	Description
tbsCertificate.version	Indicates the certificate version. Always set to v3 (value is 2)
tbsCertificate.serialNumber	Unique integer assigned by the issuing CA to the certificate
tbsCertificate.signature	OID and optional parameters defining the algorithm used to sign the certificate. This field contains the same algorithm identifier as the signatureAlgorithm field below
tbsCertificate.issuer	Distinguished Name of the CA that issued the certificate
tbsCertificate.validity	Defines when the certificate becomes active and when it expires
tbsCertificate.subject	Distinguished Name identifying the entity whose public key is certified in the subjectPublicKeyInfo field
tbsCertificate.subjectPublicKeyInfo	Field contains the public key material (public key and parameters) and the identifier of the algorithm with which the key is used
tbsCertificate.issuerUniqueID	Optional field to allow reuse of issuer names over time
tbsCertificate.subjectUnique ID	Optional field to allow reuse of subject names over time
tbsCertificate.extensions	Extension data
signatureAlgorithm	OID and optional parameters defining the algorithm used to sign the certificate. This field contains the same algorithm identifier as the signature field in tbsCertificate
signatureValue	Digital signature computed over the ASN.1 DER-encoded tbsCertificate

All certificates and CRLs described in this specification are signed with the RSA signature algorithm, using SHA-1 as the hash function. The RSA signature algorithm is described in PKCS #1 [RSA1]; SHA-1 is described in [FIPS-180-2].

III.1.1 tbsCertificate.validity.notBefore and tbsCertificate.validity.notAfter

CM certificates have a validity period greater than the operational lifetime of the CM. Manufacturer CA and the CableLabs Mfg CA certificates are valid from the issuance date for a period defined by [DOCSIS OSSI 3.0] and re-issued in a period defined by [DOCSIS OSSI 3.0]. The DOCSIS Root CA certificate is valid from the date when the DOCSIS Root CA starts operating for a period defined by the [DOCSIS OSSI 3.0] and re-issued in a period defined by [DOCSIS OSSI 3.0].

This specification assumes the operational lifetime of a CM will not exceed twenty years. The validity period of a CM certificate typically begins with the device's date of manufacture and extends for at least 20 years.

Validity periods are encoded as ASN.1 UTCTime values [X.680]. The year field (YY) is less than or equal to 50.

III.1.2 tbsCertificate.serialNumber

The serial number is a positive integer that is unique for each certificate issued by a given CA (*i.e.,* the issuer name and serial number together identify a unique certificate).

III.1.3 tbsCertificate.signature and signatureAlgorithm

All certificates and CRLs described in this specification are signed with the RSA signature algorithm, using SHA-1 as the one-way hash function. The RSA signature algorithm is described in PKCS #1 [RSA1]; SHA-1 is described in [FIPS-180-2].

The ASN.1 OID used to identify the "SHA-1 with RSA" signature algorithm is:

```
sha-1WithRSAEncryption OBJECT IDENTIFIER ::= {
iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
pkcs-1(1) 5}
```

When the sha-1WithRSAEncryption OID appears within the ASN.1 type AlgorithmIdentifier, as is the case with both tbsCertificate.signature and signatureAlgorithm, the parameters component of that type is the ASN.1 type NULL.

III.1.4 tbsCertificate.issuer and tbsCertificate.subject

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 T.61/TeletexString 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 T.61/TeletexString 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.1.4.1 DOCSIS Root CA Certificate

countryName=US
organizationName=Data Over Cable Service Interface Specifications
organizationalUnitName=Cable Modems
commonName=DOCSIS Cable Modem Root Certificate Authority

The countryName, organizationName, organizationalUnitName and commonName attributes are present and have the values shown above. No other attributes are present.

III.1.4.2 CableLabs Mfg CA Certificate

countryName=US
organizationName=CableLabs, Inc.
organizationalUnitName=DOCSIS
organizationalUnitName=D CA00001
commonName=CableLabs, Inc. Cable Modem Root Certificate

The countryName, organizationName, organizationalUnitName and commonName attributes are present and have the values shown above. The two organizationalUnitName attributes are in the order shown. No other attributes are present.

III.1.4.3 Manufacturer CA Certificate

countryName=<Country of Manufacturer>
[stateOrProvinceName=<state/province>]
[localityName=<City>]
organizationName=<Company Name>
organizationalUnitName=DOCSIS
[organizationalUnitName=<Manufacturing Location>]
commonName=<Company Name> [<Serial Identifier>] Cable Modem Root Certificate Authority [<Serial Identifier>]

Values in square brackets ([]) may or may not be present.

Values in angle brackets (<>) indicate that appropriate text as indicated is present:

<Country of Manufacturer>: two-letter country code;

<state/province>: state or province, or equivalent governmental entity.

<City>: city, metropolitan area, or equivalent entity.

<Company Name>: name that identifies the company

<Manufacturing Location>: name that identifies the location of manufacture

<Serial Identifier>: string such as “1”, “ONE”, “A”, “I”, etc., to act as an identifier.

The countryName, organizationName, DOCSIS organizationalUnitName, and commonName attributes are present and have the values shown.

The organizationalUnitName representing manufacturing location should be present. If present, it immediately follows the organizationalUnitName having value “DOCSIS.”

The stateOrProvinceName and localityName may be present.

No other attributes are present.

III.1.4.4 CM Device Certificate

countryName=<Country of Manufacturer>
organizationName=<Company Name>
organizationalUnitName=<Manufacturing Location>
commonName=<Serial Number>
commonName=<MAC Address>

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;

<Manufacturing Location>: name that identifies the location of manufacture;

<MAC Address>: MAC address of the CM.

The MAC address in the CM Certificate will be the same as the MAC address in the BPKM Attributes field.

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 organizationalUnitName should be identical to the organizationalUnitName in the issuer Name describing a manufacturing location.

The countryName, organizationName, organizationalUnitName, and commonName (<MAC Address>) attributes are present. No other attributes are present.

tbsCertificate.subjectPublicKeyInfo

The tbsCertificate.subjectPublicKeyInfo field contains the public key and the public key algorithm identifier.

The tbsCertificate.subjectPublicKeyInfo.algorithm field is an AlgorithmIdentifier structure. The AlgorithmIdentifier's algorithm is RSA encryption, identified by the following OID:

```
rsaEncryption OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840)
rsadsi(113549) pkcs(1) pkcs-1(1) 1}
```

The AlgorithmIdentifier's parameters field has the ASN.1 type NULL.

The RSA public key is encoded using the ASN.1 type RSAPublicKey:

```
RSAPublicKey ::= SEQUENCE {
        modulus             INTEGER, -- n
        publicExponent      INTEGER, -- e –
}
```

The value of the BIT STRING tbsCertificate.subjectPublicKeyInfo.subjectPublicKey is the DER encoding of the RSAPublicKey.

III.1.5 tbsCertificate.issuerUniqueID and tbsCertificate.subjectUniqueID

The issuerUniqueID and subjectUniqueID fields are absent from all DOCSIS certificates.

III.1.6 tbsCertificate.extensions

The CM Device certificate and the DOCSIS Manufacturer CA certificate may omit all extensions, even those mandated by [RFC 3280].

The CM Device certificate and the DOCSIS Manufacturer CA certificate may include extensions as described in Section III.1.6.1and Section III.1.6.2, respectively. Section III.1.6.3 specifies the CableLabs Mfg CA Certificate extensions. Section III.1.6.4 specifies the requirements on the extensions of the DOCSIS Root CA certificate. Extensions conform to [RFC 3280].

III.1.6.1 CM Device Certificates

The CM Device certificate may contain non-critical extensions; it contains no critical extensions. If the KeyUsage extension is present, the digitalSignature and keyEncipherment bits are set, the keyCertSign and cRLSign bits are unset, and all other bits should be unset A non-critical basicConstraints extension may be present.

III.1.6.2 Manufacturer CA Certificates

Manufacturer CA certificates may contain basicConstraints or keyUsage extensions. These extensions may be either critical or non-critical extensions.

Manufacturer CA certificates may contain noncritical extensions; they do not contain critical extensions other than the Basic Constraints and Key Usage extensions.

If the Key Usage extension is present in a Manufacturer CA certificate, the keyCertSign bit is set, the cRLSign bit may be set, and all other bits should be unset.

If the Basic Constraints extension is present, the cA is set to TRUE and the pathLenConstraint is set to 0.

III.1.6.3 CableLabs Mfg CA Certificate

The CableLabs CA certificate contains critical basicConstraints and keyUsage extensions. These are the only critical extensions.

In the keyUsage extension, the keyCertSign and cRLSign bits are all set. All other bits should be unset.

In the basicConstraints extension the cA is set to TRUE and the pathLenConstraint is set to 0.

The CableLabs CA certificate may contain noncritical extensions.

III.1.6.4 DOCSIS Root CA Certificate

The DOCSIS Root CA certificate contains critical basicConstraints and keyUsage extensions. These are the only critical extensions.

The DOCSIS Root CA certificate may contain noncritical extensions.

For the keyUsage extension, the keyCertSign bit is set, the cRLSign bit may be set, and all other bits should not be set.

For the basicConstraints extension, the cA is set to TRUE and the pathLenConstraint is set to 1.

III.1.7 Code Verification Certificate Format

The format used for the CVC is [X.509]-compliant. The CVC is a DER encoded structure defined in Table III–2.

The CVC contains exactly one extension: the Extended Key Usage extension. The Extended Key Usage extension is marked as critical. The extension contains a single KeyPurposeID with the value id-kp-codeSigning. If the extension is not present, or is not flagged critical, or includes any KeyPurposeID other than id-kp-codeSigning, the CM will halt the validation process and discard the CVC.

Table III–2 - DOCSIS X.509 Compliant Code Verification Certificate

X.509 Certificate Field	Description
Certificate {	
tbsCertificate	
version	v3(2)
serialNumber	Integer, size (1..20) octets
signature	SHA-1 with RSA, null parameters
issuer	
countryName	US
organizationName	Data Over Cable Service Interface Specifications

X.509 Certificate Field	Description
organizationalUnitName	Cable Modems
commonName	DOCSIS Cable Modem Root Certificate Authority
validity	
notBefore	utcTime (GMT), YYMMDDhhmmssZ
notAfter	utcTime (GMT), YYMMDDhhmmssZ
subject	
countryName	<country of subject company>
organizationName	<subject code-signing agent>
organizationalUnitName	DOCSIS
commonName	Code Verification Certificate
subjectPublicKeyInfo	
algorithm	RSA encryption, null parameters
subjectPublicKey	1024-bit, 1536-bit, or 2048-bit modulus
extensions	
extKeyUsage	
critical	true
keypurposeId	id-kp-codeSigning
signatureAlgorithm	SHA-1 with RSA, null parameters
signature Value	
} *end certificate*	

III.1.8 signatureValue

In all DOCSIS certificates, the signatureValue contains the RSA/SHA-1 signature computed over the DER-encoded tbsCertificate. The resulting signature is DER encoded as a BIT STRING and included in the certificate's signatureValue field.

III.2 Certificate Provisioning

This section provides the guidelines, and/or examples related to the Digital Certificate management process and policy.

III.2.1 DOCSIS Root CA

The DOCSIS Root CA issues two kinds of digital certificates: the CableLabs Mfg CA Certificate or Manufacturer CA Certificate (depending on "Centralized" or "Distributed" Model is used) which is embedded in the CM and verified by the CMTS in order to authenticate the CM during the provisioning process. The other certificate issued by the DOCSIS Root CA is the Manufacturer Code Verification Certificate (CVC) which is embedded in the CM Code File and verified by the CM in order to authenticate the CM Code File during Secure Software Download.

The CableLabs issued DOCSIS Root CA Certificate needs to be delivered to cable operators and/or CMTS vendors to provision the DOCSIS Root CA Certificate in the CMTS to realize the correct CM Authentication. Since the DOCSIS Root CA Certificate is also used to issue Code Verification Certificates (CVCs) it needs to be delivered to

CM vendors because the Public Key extracted from the issuing CA's Certificate is embedded in the CM for the CM to verify the CVC in the CM Code File.

III.2.2 Digital Certificate Validity Period and Re-issuance

III.2.2.1 DOCSIS Root CA Certificate

The validity period of the DOCSIS Root CA Certificate is 30 years.

III.2.2.2 DOCSIS Manufacturer CA Certificate

The DOCSIS Root CA PKI is migrating from a distributed to a centralized architecture and will no longer issue DOCSIS Manufacturer CA certificates. For all new CA requests, the vendor will need to contact CableLabs to arrange access to the CableLabs Mfg CAs for issuance of DOCSIS CM Device certificates. Vendors with existing DOCSIS Manufacturer CAs may continue to use their CAs to meet the certificate needs for specifications that allow the use of DOCSIS Manufacturer CAs. Vendors with existing DOCSIS Manufacturer CAs may opt to receive all their CM Device certificates from CableLabs Mfg CAs. However, vendors with existing DOCSIS Manufacturer CAs that have been compromised or that need to replace their existing CA will be migrated to the appropriate CableLabs Mfg CA.

III.2.2.3 DOCSIS Code Verification Certificate

When the DOCSIS Root CA newly issues the DOCSIS Manufacturer Code Verification Certificate (CVC), the following conditions apply:

- `tbsCertificate.validity.notBefore` value will be the actual issuance date and time
- `tbsCertificate.validity.notAfter` will not exceed an actual issuance date and time by 10 years, and will be valid at least 2 years from the actual issuance date.

Before the DOCSIS Manufacturer CVC expires, the certificate with the same information, except the `tbsCertificate.validity.notBefore`, the `tbsCertificate.validity.notAfter` and `tbsCertificate.serialNumber` needs to be re-issued. The CM vendors are required to obtain the re-issued DOCSIS Manufacturer CVC from the DOCSIS Root CA at least 6 months before the `tbsCertificate.validity.notAfter` value of the current DOCSIS Manufacturer CVC.

When the DOCSIS Root CA re-issues the DOCSIS Manufacturer CVC, the following attribute values will be the same as the current DOCSIS Manufacturer CVC:

- tbsCertificate.issuer
- tbsCertificate.subject

The `tbsCertificate.validity.notBefore` value will be between the `tbsCertificate.validity.notBefore` value of the current DOCSIS Manufacturer CVC, and the actual issuance date and time. (That is, the `tbsCertificate.validity.notBefore` value can be the same as the `tbsCertificate.validity.notBefore` value of the current DOCSIS Manufacturer CVC, the actual issuance date and time, or any value between the two values.) In addition, the `tbsCertificate.validity.notAfter` will be the actual re-issuance date and time plus 2 to 10 years.

The DOCSIS Root CA is responsible for registering names of authorized code-signing agents. Code-signing agents include the CM manufacturers and operators. It is the responsibility of the DOCSIS Root CA to guarantee the uniqueness of the organizationName of every code-signing agent. When assigning organizationNames to code co-signers:

- The organizationName used to identify itself as a code co-signer agent in a CVC is assigned by the DOCSIS Root CA.

- The name is a printable string of eight hexadecimal digits that uniquely distinguishes a code-signing agent.

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.2.3 CM Code File Signing Policy

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 3.0 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.2.3.1 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 [PKCS#7] signingTime value in the manufacturer's signing information. The [PKCS#7] signingTime value should be equal to or greater than the CVC's validity start time. The [PKCS#7] 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.2.3.2 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.2.4 CM Code File Format

A single file is used to encapsulate the code for the cable modem. The code file is a DOCSIS [PKCS#7] signed data message that includes:

- The Manufacturer’s Code Verification Signature (CVS);
- The Manufacturer’s Code Verification Certificate (CVC), signed by the DOCSIS Root CA;
- The code image (compatible with the destination CM) as signed content;

- Optionally, when the MSO co-signs the code file:
    a) The MSOs CVS;
    b) The MSOs CVC signed by the DOCSIS Root CA.
- Optional Root CA Public Key for the CVC verification;
- Optional Manufacturer Certificate(s);
- Optional CVC Root CA Certificate;
- Optional CVC CA Certificate.

The code file complies with [PKCS#7] and is DER encoded. The code file matches the structure shown in Appendix III. A code file example is shown in Annex D.

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 subclause 7.2.2.28. (Length is zero if there are no sub-TLVs.)
RootCAPublicKey()	Optional TLV for the Root CA Public Key for CVC Verification, formatted according to the RSA-Public-Key TLV format (Type 4) defined in the subclause 7.2.2.4.
MfgCerts()	Optional TLV for one or more DER-encoded Manufacturer CA Certificate(s) each formatted according to the CA-Certificate TLV format (Type 17) defined in the subclause 7.2.2.17.
ClabCVCRootCACert()	Optional TLV for one DER-encoded certificate formatted according to the CVC-Root-CA-Certificate TLV format (Type 51) defined in the subclause 7.2.2.29.
ClabCVCCACertificate()	Optional TLV for one DER-encoded certificate formatted according to the CVC-CA-Certificate TLV format (Type 52) defined in the subclause 7.2.2.30.
}	
CodeImage()	Upgrade code image
}	

III.2.4.1 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.2.4.1.1 Code Signing Keys

The digital signature uses the RSA Encryption Algorithm [RSA3] with SHA-1[FIPS-180-2]. The RSA key modulus is either 1024 bits, 1536 bits, or 2048 bits in length.

Table III–3 - DOCSIS PKCS#7 Signed Data

PKCS#7 Field	Description
signedData {	
Version	version = 1
digestAlgorithmIdentifiers	SHA-1
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
msoCVC()	Optional; required for cable operator co-signatures
} *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
CountryName	US
organizationName	Data Over Cable Service Interface Specifications
organizationalUnitName	Cable Modems
commonName	DOCSIS Cable Modem Root Certificate Authority
certificateSerialNumber	from CVC; Integer, size (1..20) octets
digestAlgorithm	SHA-1
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
CountryName	US
organizationName	Data Over Cable Service Interface Specifications
organizationalUnitName	Cable Modems

PKCS#7 Field	Description
commonName	DOCSIS Cable Modem Root Certificate Authority
certificateSerialNumber	from CVC; Integer, size (1..20) octets
digestAlgorithm	SHA-1
authenticatedAttributes	
contentType	data; contentType of signedContent
signingTime	UTCTime (GMT), YYMMDDhhmmssZ
messageDigest	digest of the content as defined in [PKCS#7]
digestEncryptionAlgorithm	rsaEncryption
encryptedDigest	
} *end mso signer info*	
} *end signer info*	
} *end signed data*	

III.2.4.1.2 Code Verification Certificate Format

The format used for the CVC is defined in Appendix III.1.7.

III.2.4.1.3 Code Verification Certificate Revocation

The CM is not required to support CVC revocation.

However, 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.2.4.2 Signed Content

The SignedContent field of the code file contains the CodeImage and the DownloadParameters fields, which may contain up to four items: a DOCSIS Root CA Public key, a Manufacturer CA certificate, a CVC Root CA certificate, and a CVC 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
Ben Bekele	Cox Cable
Kirk Erichsen	Adelphia
Brian Epstein	Comcast
Doc Evans	ARRIS Interactive
Mohan Gundu	BigBand Networks
Stuart Hoggan	CableLabs
Han-Seung Koo	ETRI
Oscar Marcia	CableLabs
Rick Mayberry	Comcast
Doug Nguyen	Broadcom
Niki Pantelias	Broadcom
Mike Patrick	Motorola
Lucy Pollak	Texas Instruments
Matt Schmitt	CableLabs
Nathan Vanderschaaf	Motorola
Shengyou Zeng	Cisco

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
Bill Aiello	AT&T
Burcak Beser	3com
Jeff Carr	Broadcom Corporation
Rusty Cashman	Correlant Communications
Lisa Denney	Broadcom Corporation
Jonathan Fellows	General Instrument Corporation
Stuart Green	Nortel Networks
George Hart	Rogers Cable TV Ltd.
Sasha Medvinsky	General Instrument Corporation
John Pickens	Com21
Mike Sabin	Com21
Mike St. Johns	@Home Corporation
Katherine Unger	Motorola Corporation
Wilson Sawyer	Nortel Networks

Contributor	Company Affiliation
Lior Storfer	Libit Signal Processing Ltd.
Mark Sumner	Motorola Corporation
Man Wong	Cisco Systems
James Yee	Com21
Kazuyoshi Ozawa	Toshiba

We would particularly like to thank Doc Evans (ARRIS Interactive) for being a lead author and his extensive contributions in improving the clarity of this specification. We would also like to thank Mohan Gundu (BigBand Networks), Doug Nguyen (Broadcom), and Shengyou Zeng (Cisco) for being lead authors. We thank Stuart Hoggan (CableLabs), who was the Security Focus Team Lead. And finally, many thanks go out to all the active members of the Security focus team who contributed to this specification.

Appendix V Revision History (Informative)

V.1 Engineering Changes for CM-SP-SECv3.0-I02-061222

The following Engineering Changes are incorporated into CM-SP-SECv3.0-I02-061222:

ECN	ECN Date	Summary
SECv3.0-N-06.0287-2	10/4/06	Correct CMTS requirement when EAE is completed
SECv3.0-N-06.0291-1	9/20/06	Correct spec to refer to CMTS Extended MIC instead of CMTS MIC
SECv3.0-N-06.0292-2	10/11/06	Correct SAV to be applied to multiple upstream packets on multiple SIDs
SECv3.0-N-06.0296-1	10/11/06	Isolation Packet PDU requirement
SECv3.0-N-06-0322-1	11/1/06	D3.0 Security Omnibus EC

V.2 Engineering Changes for CM-SP-SECv3.0-I03-070223

The following Engineering Changes are incorporated into CM-SP-SECv3.0-I03-070223:

ECN	ECN Date	Summary
SECv3.0-N-06.0330-1	12/13/06	Add entries to Abbreviations and Acronyms list
SECv3.0-N-06.0345-2	12/20/06	SA signaling for Multicast (SEC)

V.3 Engineering Changes for CM-SP-SECv3.0-I04-070518

The following Engineering Changes are incorporated into CM-SP-SECv3.0-I04-070518:

ECN	Date Accepted	Summary
SECv3.0-N-07.0379-1	2/14/2007	Reg-Req Message Encryption Inconsistencies
SECv3.0-N-07.0387-1	2/28/2007	BPI+ configuration file settings state machine update clarification
SECv3.0-N-07.0409-1	4/11/2007	Omnibus ECR