Society of Cable Telecommunications Engineers

ENGINEERING COMMITTEE

Digital Video Subcommittee

AMERICAN NATIONAL STANDARD

ANSI/SCTE 41 2011

POD Copy Protection System

NOTICE

The Society of Cable Telecommunications Engineers (SCTE) Standards are intended to serve the public interest by providing specifications, test methods and procedures that promote uniformity of product, interchangeability and ultimately the long term reliability of broadband communications facilities. These documents shall not in any way preclude any member or non-member of SCTE from manufacturing or selling products not conforming to such documents, nor shall the existence of such standards preclude their voluntary use by those other than SCTE members, whether used domestically or internationally.

SCTE assumes no obligations or liability whatsoever to any party who may adopt the Standards. Such adopting party assumes all risks associated with adoption of these Standards, and accepts full responsibility for any damage and/or claims arising from the adoption of such Standards.

Attention is called to the possibility that implementation of this standard may require the use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. SCTE shall not be responsible for identifying patents for which a license may be required or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

Patent holders who believe that they hold patents which are essential to the implementation of this standard have been requested to provide information about those patents and any related licensing terms and conditions. Any such declarations made before or after publication of this document are available on the SCTE web site at http://www.scte.org.

140 Philips Road
Exton, PA 19341

CONTENTS

NOTICE I

1. INTRODUCTION 1

1.1 Scope 1

1.2 References 1

1.2.1 Normative reference list 1

1.2.2 Informative Reference list 2

1.2.3 Reference acquisition 2

1.3 Acronyms, Abbreviations, and Defined Terms 4

1.4 Copy Protection System Components 6

1.5 Implementation Outline 6

1.6 Historical Perspective 7

1.7 Related Documents 7

2. SYSTEM OVERVIEW (INFORMATIVE) 8

2.1 NRSS Copy Protection Framework 8

2.2 Device Authentication 8

2.2.1 Bi-Directional Host and Cable System 9

2.2.2 Manual Return Authentication 9

2.2.3 POD Support for Multiple Hosts 9

2.3 Key Exchange and Transport Stream Protection 10

2.3.1 Setup Phase 10

2.3.2 Key Derivation Phase 10

2.3.3 Interface Encryption 11

2.4 Data Channel Protection 12

2.4.1 Copy Control Information 12

2.4.2 Rules for CP-Scrambling based on EMI value and CA-Scrambling 12

2.5 Identifying Fraudulent Devices and Disabling of Services 12

3. HOST AUTHENTICATION MECHANISMS 13

3.1 Protocol Components 13

3.1.1 X.509 Version 3 Certificate 13

3.1.2 Device Parameters 13

3.1.3 System Parameters 14

3.1.4 Processing Basics 14

3.2 POD/Host Binding and Registration 18
3.2.1 ID ReportING Mechanism 19
3.2.2 Authentication Phase 1 – Certificate Verification & DH Key Exchange 20
3.2.3 Authentication Phase 2 – Authentication Key Verification 21
3.2.4 Authentication Phase 3 – Headend Report Back 21
3.2.5 Headend Report Back Methods 22
3.3 Power-up Re-Authentication 25
3.4 POD Operation with Multiple Hosts 25
3.5 Host Operation with Multiple PODs 25

4. CRYPTOGRAPHIC FUNCTIONS 26
4.1 Authentication Key Generation 26
4.2 Copy Protection Key Generation 27
4.2.1 Basic Key Generation Protocol 27
4.2.2 POD Module Copy Protection Key 28
4.2.3 Host Copy Protection Key 29
4.3 CP-Key Refresh 29
4.3.1 Key Session Period 30
4.3.2 Key Refresh Period 30
4.3.3 CA System Key Refresh 32
4.3.4 Key Refresh Initialization 32
4.3.5 Channel Change 32
4.3.6 Two Key Synchronization Mode (Informative) 33
4.3.7 Transport Scrambling Control Field 33
4.4 Diffie-Hellman Key Exchange Algorithm 34
4.4.1 Algorithm Overview 34
4.4.2 Algorithm Implementation 35
4.5 SHA-1 Secure Hash Algorithm 36
4.6 Random Number Generation 36
4.7 DFAST Algorithm 37
4.7.1 Algorithm Overview 37
4.7.2 DFAST Characteristics 37
4.8 RSA Digital Signatures 37

5. HOST SERVICE REVOCATION MECHANISMS 38
5.1 System Issues 38
5.2 Revocation Circumstances [Informative] 38
5.3 Fraudulent Host Identification 38
5.4 CA System Revocation & Selective Denial of Services 38

5.4.1 Definition of Revocation 38
5.4.2 Selective Service Denial 38
5.5 The Revocation Process 39
5.6 Implementation in the Headend 40
6. COPY CONTROL INFORMATION (CCI) 41
6.1 CCI Definition 41
6.1.1 EMI - Digital Copy Control Bits 41
6.1.2 APS - Analog Protection System 42
6.2 Associating CCI with a Service 42
6.3 Conveying CCI from Headend to POD 42
6.4 Conveying CCI from POD to Host 42
6.4.1 CCI Delivery Instances 43
6.4.2 Authenticated Tunnel Protocol 44
7. TRANSPORT SCRAMBLING POD TO HOST 45
7.1 MPEG Scrambling 45
7.1.1 Scrambling Rules 45
7.2 Transport Processing 46
7.3 Timing of Scrambling Mode Transitions 46
7.4 CP-Scrambling as a function of CA-Scrambling and EMI Value 46
8. HOST, POD, & HEADEND MESSAGING PROTOCOLS 47
8.1 Message Protocol Overview 47
8.2 POD & Host Common Messages 48
8.2.1 Opening a Session 48
8.2.2 Host Capability Evaluation 50
8.2.3 Copy Protection Key Generation 52
8.2.4 Host and POD Synchronization 55
8.3 POD-Host CP Message Protocol 57
8.3.1 Protocol Flow Overview 57
8.3.2 Host Authentication Messages 60
8.3.3 Host Authentication Key Verification Messages 63
8.4 CCI Simple Authentication Tunnel Protocol (SATP) Messages 64
APPENDIX A. LUHN CHECK DIGIT (NORMATIVE) ..... A
APPENDIX B. APPLYING CP-KEY TO DES ENGINE (NORMATIVE) ..... B

Method of Application ..... **B**

Examples of CP encryption oF MPEG DATA IN transport packets ..... **C**

List of Figures

Figure 3.1-A POD-Host CP Operation 17
Figure 3.1-B Headend CP Operation [informative] 18
Figure 3.2-A Example ID Reporting Screen 20
Figure 3.2-B Example CP System Failure Notification message 21
Figure 4.0-A Overall POD Copy Protection 26
Figure 4.3-A POD CP-Key Refresh Session Flow Chart 31
Figure 4.4-A Diffie-Hellman Key Agreement Protocol between POD and Host 34
Figure 6.4-A CCI Delivery Sequence 43
Figure 8.1-A Copy Protection Message Protocol Overview 47
Figure 8.3-A POD-Host CP Message Protocol Overview 57

List of Tables

Table 3.1-A Length of Device Parameters in the Host Authentication 13
Table 3.1-B Length of System Parameters 14
Table 4.2-A Length of Keys and Parameters Used in the Key Generation 29
Table 4.3-A MPEG Transport_scrambling_control values 33
Table 5.6-A CRL Based Host and POD Service Revocation 40
Table 6.1-A CCI Bit Assignments 41
Table 6.1-B EMI Values and Content 41
Table 6.1-C APS Value Definitions 42
Table 7.4-A CP-Scrambling based on CA-Scrambled State and EMI Value 46
Table 8.2-A Copy Protection Open Session Information 48
Table 8.2-B Open_Session_Request() Message Syntax 49
Table 8.2-C Copy Protection Resource Class 49
Table 8.2-D Open_Session_Response() Message Syntax 50
Table 8.2-E Host CP Support Capability Evaluation Messages 50
Table 8.2-F CP_open_req() Message Syntax 51
Table 8.2-G CP_open_cnf() Message Syntax 51
Table 8.2-H CP_system_id_bitmask Values 51
Table 8.2-I CP_data in the Transmission Key Generation Messages 52
Table 8.2-J CP_data_req() Message Syntax In the Key Generation Messages 52
Table 8.2-K Datatype_ID and Datatype_length Values 53
Table 8.2-L CP_system_id Values 54
Table 8.2-M CP_data_cnf() Message Syntax In the Key Generation Messages 54
Table 8.2-N Host and POD module Synchronization Messages 55
Table 8.2-O CP_sync_req() Message Syntax 55
Table 8.2-P CP_sync_cnf() Message Syntax 55
Table 8.2-Q Status_field Value 55
Table 8.3-A POD-Host CP Message Reference Sections 58
Table 8.3-B Host Authentication Messages 60
Table 8.3-C CP_data_req in the Host Authentication Request Message 61
Table 8.3-D CP_data_cnf in the Host Authentication Response Message 62
Table 8.3-E Host Authentication Key Verification Messages 63
Table 8.3-F CP_data_req in the Authentication Key Verification Request Message 63
Table 8.3-G CP_data_cnf in the Authentication Key Verification Response Message 64
Table 8.4-A CCI Simple Authentication Tunnel Protocol Messages 64
Table 8.4-B CP_data_req() Message Syntax in SATP Key Generation 66
Table 8.4-C CP_data_cnf() Message Syntax in CCI SATP Key Generation 66
Table 8.4-D CP_data_req() Message Syntax in CCI SATP Transmission 67
Table 8.4-E CP_data_cnf() Message Syntax in CCI SATP Transmission 67

1. INTRODUCTION

1.1 SCOPE

In digital Cable systems, high value movies and video programs ("content") are protected by a conditional access scrambling system. A properly authorized CableCARD™ Point of Deployment (POD) security module removes the scrambling and, based on the Content Control Information from the Headend, may rescramble the content before delivering it to consumer receivers and set-top terminals ("Host devices") across the POD-Host interface defined in ANSI/SCTE 28 2007.

This standard defines the characteristics and normative specifications for the system that prevents unrestricted copying of such high value content as it crosses the POD-Host interface.

Content that is delivered unscrambled over Cable systems is not subject to this standard. Indeed, this standard would not provide any protection against unrestricted copying of such content. Any unscrambled content output by the Host on the POD interface will not benefit by scrambling upon its subsequent output from the POD on that same interface.

This standard provides methods for authenticating Host devices, for binding POD modules to Host devices including Diffie-Hellman key exchange, for copy protection key generation, for rescrambling high value content to protect against unauthorized copying (after the POD module employs the conditional access system to descramble it) and then descrambling by the Host, and for transmission and authentication of Copy Control Information. It also provides for revocation of Host devices that are determined to be fraudulent or non-compliant.

This standard requires the use of technology that must be licensed from CableLabs. The technology is called DFAST (U.S. Patent No. 4,860,353 and related know-how), and the license is PHILA – "POD-Host Interface License Agreement", available from CableLabs. Please refer to section 1.2.3 under "DFAST Technology, PHILA, and PHICA" for contact information for such a license.

1.2 REFERENCES

1.2.1 NORMATIVE REFERENCE LIST

The following standards contain provisions that, through reference in this text, constitute normative provisions of this Specification. At the time of publication, the editions indicated are current. All standards are subject to revision, and parties to agreements based on this Specification are encouraged to investigate the possibility of applying for the most recent editions of the standards listed in this section.

1. ANSI/SCTE 28 2007: "Host-POD Interface Standard"

2. CEA-679-C: "National Renewable Security Standard," Part B, (October 2005) "National Renewable Security Standard" (March 2000)

3. ITU-T Recommendation X.509, "Information Technology – Open Systems Interconnection – The Directory: Public-key and Attribute Certificate Frameworks", August 2005.

4. FIPS PUB 46-3: "Data Encryption Standard (DES)" 1999 October 25

5. FIPS-PUB 81: "DES Modes of Operation" 1980 December 21

6. FIPS PUB 140-2: “Security Requirements for Cryptographic Modules”, 2001 May 25

7. FIPS PUB 180-2: “Secure Hash Standard”, 2002 August 1

8. FIPS PUB 186-2, “Digital Signature Standard” Federal Information Processing Standards Publications (FIPS PUB), 2000 January 27.

9. RSA1, “PKCS #1: RSA Encryption Standard”, Version 1.5, November 1993

10. MPEG: ISO/IEC 13818-1:2008 Information Technology - Generic coding of moving pictures and associated audio information: Systems

11. DFAST: Dynamic Feedback Arrangement Scrambling Technique; a component of the encryption algorithm specified by the PHICA. Available under license from CableLabs, see Section 1.1.

1.2.2 INFORMATIVE REFERENCE LIST

The following references contain information that is useful in understanding of this Specification. Some of these documents are drafts of standards or balloted standards with unresolved comments.

1. RSA3, “PKCS #10 V1.7: Certification Request Syntax Standard”, May 2000

1.2.3 REFERENCE ACQUISITION

DFAST Technology, PHILA, and PHICA: Cable Television Laboratories, Inc.

Cable Television Laboratories, Inc., 400 Centennial Parkway, Louisville, CO 80027; Telephone: 303-661-9100; Facsimile: 303-661-9199;
E-mail: opencable@cablelabs.com; URL: <www.cablelabs.com>

CEA Standards: Consumer Electronics Association

Global Engineering Documents, World Headquarters, 15 Inverness Way East, Englewood, CO USA 80112-5776; Telephone 800-854-7179; Facsimile: 303-397-2740; E-mail: global@ihs.commailto:global@his.com ; URL: <http://global.ihs.com>

IEEE Standards: Institute of Electrical and Electronic Engineers

Institute of Electrical and Electronic Engineers, 445 Hose Lane, Piscataway, NJ 08855-1331, USA; E-mail: customer.service@ieee.org ;
URL: http://standards.ieee.org/index.html

ISO: International Standards Organization

Global Engineering Documents, World Headquarters, 15 inverness Way East, Englewood, CO 80112-5776, USA: Telephone: 800-854-7179; E-mail: global@ihs.com; URL: http://global.his.com

ITU-T: International Telecommunications Union – Telecom Standardization

InternationalTelecommunicationsUnion,Geneva,Switzerland.
URL: <http://www.itu.int/publications/index.html>

NIST Publications: National Institute of Standards and Technology

National Technical Information Service (NTIS), U. S. Department of Commerce,
5285 Port Royal Road, Springfield, VA 22161;
Telephone: 1-800-553-NTIS (6847) or 703-605-6000; FAX: 703-321-8547;
E-mail orders: orders@ntis.fedworld.gov, URL: <http://www.itl.nist.gov/fipspubs/>

RSA Security

RSA Security, Inc, 174 Middlesex Turnpike, Bedford, MA 01730; Telephone: 781 515 5000; FAX: 781 515 5010; URL: <http://www.rsasecurity.com/rsalabs/pkcs>

SCTE Standards: Society of Cable Telcommunications Engineers

Society of Cable Telcommunications Engineers, 140 Philips Road, Exton, PA 19341, USA. Telphone: 610-363-6888; Facsimile: 610-363-5898; E-mail: scte@scte.org;

URL: <http://www.scte.org/standards/standardsavailable.html>

1.3 ACRONYMS, ABBREVIATIONS, AND DEFINED TERMS

APDU	Application Protocol Data Unit: a command, query, and reply message exchange protocol between POD and Host
APS	Analog Protection System for copy control of analog output video
AuthKey	Authentication Key, calculated by both the POD and Host as part of the Host authentication process
Authentication	A procedure to securely confirm that a Host or POD has a genuine X.509 certificate and that the certificate has not been revoked. Also: a means to securely confirm that a message originated in a trusted source.
Authenticated	A quality resulting from the application of an Authentication procedure; securely confirmed.
CA, CA System	Conditional Access, Conditional Access System – secures delivery of Cable services to the POD from unauthorized access
CA-only	The POD mode of CA-descrambling EMI=0 content and returning it to the Host CP-unscrambled
Cable	The Cable Television industry, services, systems, or equipment
CCI	Copy Control Information
CP	Copy Protection
CP-Key	The Copy Protection Key derived between the POD and Host, and used by the POD to CP-scramble protected content sent to the Host
CP System	The Copy Protection System described in this specification
CRL	Certificate Revocation List: the means of reporting bad Host_ID's to Cable Headends
CSR	Customer Service Representative
DES	Data Encryption Standard
DES-ECB	Data Encryption Standard – Electronic Code Book
DFAST	Dynamic Feedback Arrangement Scrambling Technique, a component of the encryption algorithm
DH	Diffie-Hellman, a public key agreement protocol based on the intractability of taking discrete logarithms over the integer field
EMI	“Encryption Mode Indicator” As used in this document the meaning of this acronym is "Copy Control" Mode Indicator for digital outputs.

EMM Entitlement Management Message

Encrypted Data modified to prevent unauthorized access (compare with "scrambled")

FAT Forward Application Transport, the 6 MHz digital channels from Headend to home and between POD and Host

Host Certificate The unique X.509 certificate issued to each Host device and used for Host authentication. Parameter name: *Host_DevCert*

Host Certificate List The pair of X.509 certificates provided by the Host to the POD for certificate verification: *Host_DevCert* and *Host_ManCert*

Host_ID The Host device's unique identification number

LSB Least Significant Bit, of a specified binary value

Manufacturer XCA X.509 certificate authority authorized by PHICA to issue POD or Host Certificates. PHICA issues each Manufacturer XCA a unique manufacturer's certificate, *POD_ManCert* or *Host_ManCert*, which are used to verify POD or Host Certificates.

MMI Man Machine Interface

MPEG The ISO/IEC 13818 specifications and ISO/IEC 13818-1 in particular

MSB Most Significant Bit, of a specified binary value

Nonce A random value generated fresh for each use and included in some Host-POD exchanges to make each exchange unique

Pass-through The POD mode of passing CA-scrambled back to the Host unchanged, leaving it unusable by the Host

PHI POD-Host Interface as specified in ANSI/SCTE 28 2007

PHICA POD-Host Interface Certificate Authority, root X.509 certificate administrator for X.509 certificates on the PHI. Identified under PHILA

PHILA POD Host Interface Licensing Agreement, covers the DFAST technology and specifies the PHI Certificate Authority - PHICA

POD or CableCARD Point of Deployment module, the removable PC-Card form factor Cable
POD Module security module

POD Certificate The unique X.509 certificate issued to each POD and used for POD authentication. Parameter name: *POD_DevCert.*

POD Certificate List The pair of X.509 certificates provided by the POD to the Host for certificate verification: *POD_DevCert* and *POD_ManCert*

POD-CP	POD copy protection, as specified in this document
POD CPS	The POD Copy Protection System, as specified in this document
POD_ID	The POD module's unique identification number
RDC	Return Data Channel: transmitted from home to Headend
Report-back	The action or process of reporting information from the POD or Host back to the Headend
Rescramble	The POD mode of CA-descrambling and CP-scrambling content
RSA algorithm	An RSA Security defined commercial public key cryptographic algorithm
SATP	Simple Authenticated Tunnel Protocol
Scrambled	Content modified to prevent unauthorized access (compare with "encrypted")
SHA-1	Secure Hash Algorithm, a cryptographic compression function
Shall	"Shall" denotes a mandatory provision of this standard
Should, May	“Should" denotes a provision that is recommended but not mandatory. "May" denotes a feature whose presence does not preclude compliance and may or may not be present at the option of the implementer.
SPDU	Session Protocol Data Unit (SPDU)
SSK	A shared secret system parameter used by both POD (SSK_P) and Host (SSK_H) to authenticate the exchange of Diffie-Hellman public key parameters.
Validation	The process of reporting the Host_ID to the system operator, checking it against a revocation list, reporting the validated Host_ID back to the POD, and the POD confirming it matches the stored Host_ID.
X.509	The ITU-T Recommendation X.509 standard
XCA	X.509 certificate authority

1.4 COPY PROTECTION SYSTEM COMPONENTS

This copy protection system (CP System) includes:

- The Cable navigation device, or Host
- The POD module
- The Cable Headend (which revokes selected services from compromised Hosts)
- The PHICA which issues device ID’s and generates X.509 manufacturer certificates.

1.5 IMPLEMENTATION OUTLINE

The POD CP System consists of the following two operational elements:

Host Authentication, based on the exchange of Host and POD certificates across the POD-Host interface. Each device verifies the other's certificate using signature verification techniques, and the Host and POD IDs are reported to the Headend. The Headend compares the IDs against a revocation list and takes appropriate revocation action against compromised devices.

Copy Protection Key Derivation, using a Diffie-Hellman shared secret key that was computed during the Host Authentication process. CP-scrambling by the POD of content marked with non-zero EMI. The POD first CA-descrambles this content and then rescrambles such High Value content using the Copy Protection Key before delivery to the Host. A companion CP-descrambling process occurs in the Host.

1.6 HISTORICAL PERSPECTIVE

This specification has its origins in EIA-679 (now CEA-679,) the National Renewable Security Standard, which was initially adopted in September 1998. Part B of that standard has the physical size, shape and connector of the computer industry PCMCIA card.

That standard did not take into account the requirements of the movie industry to protect against the unrestricted copying of digital video movies and programs.

Further extensions and modifications of EIA-679 led to the adoption of EIA-679-B in March 2000. EIA -679-B permits the use of copy protection techniques but does not select any single approach.

Consequently, the Cable industry selected the particular approach embodied in this document, and submitted it as DVS/213 in June 1999. Extensive revisions were developed by the Cable industry, and submitted as DVS 301 (January 2000). Work on this document by the Cable industry proceeded during the first half of 2000, leading to substantial changes that were embodied in DVS 301r1, r2 and finally DVS301r3 which was successfully balloted and adopted as ANSI/SCTE 41 2001.

A subsequent update was proposed, balloted and approved as SCTE 41 2003. This document updates that work to current industry practice, notably by revising the authentication method to one based on X.509 standard certificates.

1.7 RELATED DOCUMENTS

This document is intended to supplement the functionality of the POD module interface described in ANSI/SCTE 28 2007 [1] which defines how copy protection fits into the overall POD module interface functionality.

2. SYSTEM OVERVIEW (INFORMATIVE)

2.1 NRSS COPY PROTECTION FRAMEWORK

This document is based on the copy protection framework defined in CEA-679-C, National Renewable Security Standard (NRSS), Part B, section 8.9 [2] with some substantial changes. The NRSS Copy Protection Framework includes:

- POD/Host Binding,
- DES-ECB Scrambling System (FAT Channel),
- MPEG Systems for the transport layer (FAT Channel),
- NRSS-B Interface Copy Protection Resource (Data Channel),
- NRSS-B Interface Messages (Data Channel).

But, NRSS doesn't reference any of the following operations that are described in this document:

- Device authentication,
- Host revocation via selective service denial.

Under the NRSS Copy Protection Framework, the POD module checks the ability of the Host to support POD-CP by checking availability of the Copy Protection Resource as defined in ANSI/SCTE 28 2007 and verifies the Host. If the Host is not valid, the POD module goes to "pass-through" mode, simply passing any received transport stream back to the Host unchanged.

2.2 DEVICE AUTHENTICATION

The CP System requires authentication of the Host and POD prior to the POD descrambling any of the copy-protected material. The POD requests the Host's X.509 Certificate List and the Host provides it. The Host requests the POD's X.509 Certificate List and the POD provides it. Authentication is based on:

- POD being able to verify the signature of the X.509 host device certificate that contains Host_ID and the POD being able to verify the signature of the Host Manufacturer's XCA certificate; and
- Host being able to verify the signature of the X.509 POD device certificate that contains the POD_ID and the Host being able to verify the signature of the POD Manufacturer's XCA certificate.
- POD and Host proving they each hold the private key paired with the public key embedded in the X.509 certificate by signing a DH session key and sending it to the other device for signature verification.
- POD and Host proving that they can derive the authentication key.
- The Host_ID and POD_ID extracted from the X.509 certificates are not included in the CRL, as checked in the Headend.

Additionally, the POD will validate the authenticity of the X.509 Host Certificate and the Host will validate the authenticity of the X.509 POD Certificate using the License Administrator's public verification keys that are delivered by the CA System to the POD and Host in an authenticated manner. If the *Validated_Host_ID* value is the same as the Host_ID in the authenticated X.509 Host Device Certificate and the *Validated_POD_ID* value is the same as the POD_ID in the authenticated X.509 POD Device Certificate, the Host and POD can continue with the key exchange process described in Chapter 3.

2.2.1 BI-DIRECTIONAL HOST AND CABLE SYSTEM

If both the Host and Cable system support automatic report-back, via either telco modem or an on-cable return channel, the POD sends the POD and Host information on that channel without the need for manual reporting.

2.2.2 MANUAL RETURN AUTHENTICATION

The POD and Host ID's must be reported to the service provider before the Host can access Copy Protected content. The end-user or the retailer may perform this service.

For one-way Cable systems, unidirectional Hosts, or any system without a functioning automatic report-back mechanism, reporting of the authentication ID's is accomplished manually. For example: the user making a telephone call to the service provider. The manual return authentication process is detailed in section 3.2.1.

2.2.3 POD SUPPORT FOR MULTIPLE HOSTS

Although it may be technically feasible to build a POD that is capable of binding to multiple Hosts, such that the POD can be moved from Host to Host, such operation is beyond the scope of this specification. Multiple Host capability may be added to a future extension of this specification.

2.3 KEY EXCHANGE AND TRANSPORT STREAM PROTECTION

The copy protection mechanism itself consists of three phases: Setup, Key Derivation, and Interface Encryption. PODs and Hosts contain the algorithms for Diffie-Hellman (DH) key negotiation, SHA-1 hashing, and DES. The DH system-wide constants (g and n) are chosen to be large enough to be secure for the long term (e.g., n is 1024 bits for DH based on discrete log). These constants are provided under license by the PHICA. PODs and Hosts also contain private keys (x and y, respectively) and the corresponding public keys (g^x mod n and g^y mod n). The private keys x and yh are pseudorandom integers generated each time a POD and Host are bound. The private keys must also be large enough to be secure for the long term (e.g., 1024 bits for DH based on discrete log).

2.3.1 SETUP PHASE

In case of successful mutual authentication between POD and Host, the POD and the Host negotiate and derive two long term shared secrets using Discrete Logarithm Diffie-Hellman in a protocol that is detailed later in Chapter 3 to exchange data across the interface. One of the long-term secrets is a 160-bit authentication key, and the other is a 1024-bit shared DH secret value. Finally SHA-1 and DFAST intellectual property are used to generate a Copy Protection Key (CP-Key).

2.3.1.1 AUTHENTICATION KEY SETUP

Normally an authentication key ($AuthKey_P$/$AuthKey_H$) is only calculated once during POD-Host binding and stored in non-volatile memory.

If $AuthKey_H$ delivered by the Host after power-up does not match the $AuthKey_P$ stored by the POD, then the entire Host authentication process is re-initialized.

The AuthKey generation process is detailed in section 4.1.

2.3.1.2 LONG TERM DIFFIE-HELLMAN SHARED KEY VALUE

Similar to the authentication key process, Diffie-Hellman shared key calculation is normally conducted when the Host authentication process is initiated. The derived DH shared secret (DHKey) is saved in non-volatile memory once it is calculated.

2.3.2 KEY DERIVATION PHASE

After a POD module and Host complete authentication, each device derives the Copy Protection Key (CP-Key). The CP-Key is calculated based on the long-term keys, AuthKey, DH keys, and the random numbers exchanged for key generation. This CP-Key is unique to the particular POD-Host pair, and serves to bind them together for content protection. Details are discussed in chapters 3 and 4.

2.3.3 INTERFACE ENCRYPTION

The POD uses encryption techniques to scramble copy-protected content transmitted across its interface with the Host. The POD rescrambles the content it has descrambled under the conditional access system using DES in ECB mode and the computed Copy Protection Key before sending it across the interface to the Host.

The POD passes content in one of four modes determined by CA System scrambling mode and EMI values as detailed in Table 7.4-A:

- Clear: no change of CA-unscrambled and EMI=0 content which remains 'in-the-clear'
- CA-only: descrambles CA-scrambled content marked EMI=0 for output 'in-the-clear'
- Rescramble: CA-descrambles and CP-scrambles content marked EMI>0
- Pass-through: no change of CA-scrambled content (leaving it unrecognizable to the Host)

The Copy Protection Key is applied across all rescrambled packets. Only one copy-protected MPEG program is delivered from the POD to the Host at any time.

The POD CP-scrambles only the payload portion of transport packets using DES electronic code book (DES-ECB) encryption. DES-ECB encrypts data in 8-byte blocks. The ECB DES is applied block-by-block, starting at the beginning of the transport packet payload. Any short block will therefore occur at the end of the transport packet. Partial blocks (e.g. less than 8 bytes) remaining at the end of a scrambled transport packet are not encrypted. The transport packet header and adaptation field (if any) is not scrambled. See chapter 7.

2.4 DATA CHANNEL PROTECTION

There are currently no messages defined on the data channel or extended channel POD↔Host interface that require message encryption. However, CCI information travelling from POD to Host is authenticated, as described in Chapter 6.

2.4.1 COPY CONTROL INFORMATION

Copy control information (CCI) is passed from the POD module to the Host across the data channel to inform the Host device of the level of copy protection required. The CCI is sent in the clear to the Host device, but the integrity of the information is maintained by authenticating the CCI using a simple protocol.

The one-byte CCI field contains information that the Host uses to control copying of content. Two EMI bits control copying on Host digital outputs, two APS bits control copying on analog outputs, and four bits are reserved.

2.4.2 RULES FOR CP-SCRAMBLING BASED ON EMI VALUE AND CA-SCRAMBLING

1. CP-scrambling is applied only to protected content, as indicated by EMI greater than zero.
2. All copy-protected content is delivered CA-scrambled to the POD.
3. Non-copy-protected content, with EMI = 00, may be delivered to the POD either CA-scrambled or CA-unscrambled.

2.5 IDENTIFYING FRAUDULENT DEVICES AND DISABLING OF SERVICES

The POD requests and verifies the Host Certificate. The POD then extracts the Host ID and passes it, by either manual or automated means to the service provider. The service provider confirms the Host is permitted to receive protected content by returning a Host_ID validation massage back to the POD.

If the service provider determines that a fraudulent Host device should no longer receive a service or group of services, the conditional access system denies the appropriate services to the POD that is bound to the fraudulent Host. Restoration of services is also executed by the conditional access system. This CA System-based revocation is known as *Selective Service Denial* that may entail:

- Complete loss of service
- Loss of particular channels, e.g., HBO™ or Showtime™.
- Loss of individual programs on channels

The service provider may use CA mechanisms to deliver “pass/no pass” commands to the POD for each service as well as new copy protection parameters. Service provider involvement and control provides a highly flexible system for protection of content copyrights.

3. HOST AUTHENTICATION MECHANISMS

3.1 PROTOCOL COMPONENTS

Host authentication is based on:

- The POD and Host verifying the opposite device's certificate signature
- The POD and Host verifying, with the received public key, a unique message signed with the opposite device's private key.
- The CA system confirming that the POD and Host ID's extracted from the X.509 certificates are not in the CRL

3.1.1 X.509 VERSION 3 CERTIFICATE

The POD and Host each have a private key used for digital signatures and an X.509 Certificate. The certificate includes a unique ID and the public key provided to other devices to validate the digital signatures. Each device also has its Manufacturer CA's certificate that is used to sign the device certificate, and the root certificates that is used to sign the Manufacturer CA's certificate

3.1.2 DEVICE PARAMETERS

The following device parameters are used in this authentication protocol:

- DH_pubKey_H: The Host Diffie-Hellman public key, also used as a Host-generated nonce in the calculation of the Authentication Key.
- DH_pubKey_P: The POD Diffie-Hellman public key, also used as a POD-generated nonce in the calculation of the Authentication Key.
- $AuthKey_H$ (derived): The Host Authentication Key.
- $AuthKey_P$ (derived): The POD Authentication Key.

Table 3.1-A Length of Device Parameters in the Host Authentication

Key or Variables	Size (bits)
Diffie-Hellman Public Keys (DH_pubKey_H, DH_pubKey_P)	1024 each
Authentication Keys ($AuthKey_H$, $AuthKey_P$)	160 each

3.1.3 SYSTEM PARAMETERS

Table 3.1-B defines system parameter length and source:

Table 3.1-B Length of System Parameters

Key or Variable	Size (bits)	Source of Parameter
POD_ID	64 bits	PHICA and manufacturer
Host_ID	40 bits	PHICA and manufacturer
Diffie-Hellman prime (n)	1024 bits	PHICA
Diffie-Hellman base (g)	1024 bits	PHICA
RSA public signing key exponent	40 bits	PHICA

3.1.4 PROCESSING BASICS

The POD Copy Protection System (CPS) comprises the following basic steps:

1. The Host shall report copy protection as a resource during the profile inquiry process (see ref. 1). Failure to do so constitutes a failure of the copy protection system (see section 3.2.2). After the copy protection resource has been reported, the POD module shall permit CA decryption of programs with a EMI value of 00.
2. The POD module shall open a session to the copy protection resource, section 8.2.1. Failure to open a valid session constitutes a failure of the copy protection system (see section 3.2.2).
3. The POD module shall send a *CP_open_req APDU* to the Host, section 8.2.2.1.
4. The Host shall respond with *the CP_open_cnf APDU* within 5 seconds. Failure to respond to any request within 5 seconds constitutes a failure of the Host and shall cause the POD module to set the IIR flag (see ref. 1).
5. The Host shall respond with the System 2 bit set in *CP_system_id_bitmask*, section 8.2.2.2. Failure to do so constitutes a failure of the copy protection system (see section 3.2.2).
6. If the POD module contains a valid Authentication key in its non-volatile memory, it shall request the Host to send its $AuthKey_H$.
7. The Host shall respond with its $AuthKey_H$, if available. If it is not available, then it shall transmit a value of all 0's. A value of all 0's shall be recognized by the POD as an invalid $AuthKey_H$.
8. The POD module shall compare the received $AuthKey_H$ with its stored $AuthKey_P$. If the authentication keys match, then the certificates are considered valid, the DH Key and authentication keys are preserved, only the Copy Protection Key is regenerated, and continue with step 17. If the authentication keys do not match the POD shall set Headend Validated to false in nonvolatile memory and the POD and Host shall continue with step 9.
9. The POD module shall send its POD Certificate Data (*POD_DevCert* and *POD ManCert*), the newly generated DH public key (DH_pubKey_P), and the Diffie-Hellman key signature. In this message, the POD also requests that the Host deliver its Host Certificate Data (*Host_DevCert* and *Host_ManCert*) and signed DH public key (DH_pubKey_H).

10. The Host shall reply to the POD module request with its Host Certificate Data (*Host_DevCert* and *Host_ManCert*) and newly generated and signed DH public key.

11. The POD module shall verify the *Host_DevCert*, the *Host_ManCert*, the signature on the Diffie-Hellman public key (DH_pubKey_H) and extract the Host_ID from the Host certificate. If verification fails, this constitutes a failure of the copy protection system (see section 3.2.2).

12. The Host module shall verify the *POD_DevCert*, the *POD_ManCert*, the signature on the Diffie-Hellman public key (DH_pubKey_P) and extract the POD_ID from the POD certificate. If verification fails, this constitutes a failure of the copy protection system (see section 3.2.2).

13. After these exchanges, both the POD module and the Host come up with their respective authentication key, $AuthKey_P$ and $AuthKey_H$.

14. The POD module shall request $AuthKey_H$ and compare it to $AuthKey_P$. If they are not equal, this constitutes a failure of the copy protection system (see section 3.2.2). If they are equal, then the POD module and Host shall complete the Diffie-Hellman operation and store the derived authentication key into non-volatile memory.

15. If headend validation of the POD and Host ID is complete, go to step 18. Otherwise, the following depends upon what path is available to report device IDs to the headend.

    a. If possible, i.e., in a system with an active return data channel or telco return path, the POD module shall send the POD_ID and Host_ID to the cable headend via an upstream OOB private CA message or telco modem. The POD module also stores the Host_ID and POD_IDs in nonvolatile memory so they can be compared with the validated IDs received back from the headend. Status of this transmission shall be stored in non-volatile memory to prevent unnecessary retranmissions.

    b. In systems in which no automated return path is available, the POD module sends a display message as defined in section 3.2.5.1.

16. The headend CA System records the pairing between Host_ID and POD_ID, and looks for the Host_ID in the revocation list. If not found, the CA System re-sends the Host_ID and POD_ID back to the POD module in a private authenticated CA System ID validation message. This message may be sent to the POD module substantially later in time than when the POD and Host_ID's are reported to the heandend.

17. Until the POD and Host ID headend validation is completed the POD module shall CA decrypt onlythose services with an EMI value of 00.

[Asynchronously and independent of this process, the CA system typically sends an EMM to the POD module to authorize appropriate services. Some of these services may have EMI values of 01, 10, or 11. The POD module shall not CA-descramble services with these EMI values until headend ID validation is complete, even if these services are otherwise authorized in an EMM.]

18. The POD module authenticates the ID validation message and compares the received Host_ID and POD_ID with the ID's extracted from the Host device certificate and the POD device certificate, respectively. If they match, the POD module shall store the Headend Validated True in nonvolatile memory and allow CA decryption of high value content with EMI values of 01, 10, or 11, if so authorized by an EMM. If they do not match, the POD module shall continue to limit its CA decryption to those services with EMI values of 00 only.

19. If the headend CA system receives a new revocation list, it shall examine all previously reported IDs and if there are any matches, it shall notify the cable operator.

20. If a POD module reports a failure of the copy protection system to the headend CA system, the headend CA system shall notify the cable operator.

The following flowcharts show an implementation of the above steps:

POD-Host CP Operation

Figure 3.1-A POD-Host CP Operation

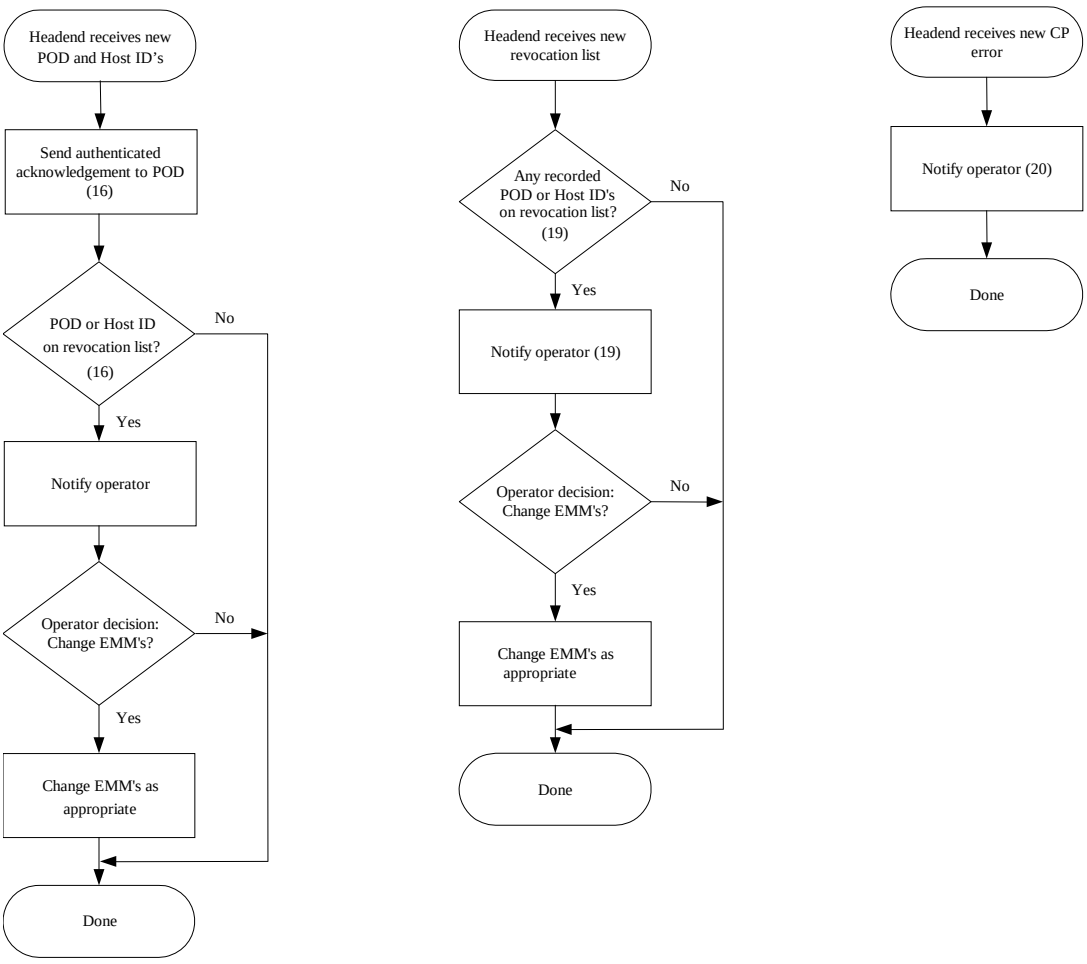

Figure 3.1-B Headend CP Operation [informative]

3.2 POD/HOST BINDING AND REGISTRATION

After POD insertion and a complete PCMCIA power up initialization, the Host authentication protocol below is conducted between the POD and Host. At this initial binding, the process of Host authentication with the POD module has three steps:

- certificate verification
- derived Authentication Key step
- Host_ID and POD_ID report-back to the Headend, Headend validation

The Certificate Revocation List (CRL) is used in the Headend to validate the POD and Host as part of the revocation through the CA System (not in the POD or Host).

The CA System shall have the ability to command the POD to Full Copy Protection Reinitialization, as if the POD were inserted into its Host for the first time. When this occurs, the POD shall begin Authentication all over again, as if it had never before been registered or authenticated with that specific

Host. For one-way cable systems or Uni-Directional Hosts this will require the consumer to call the operator to report the Host and POD Module IDs for validation.

3.2.1 ID REPORTING MECHANISM

The Host_ID and POD_ID must be reported to the service provider before the POD will provide High Value content to the Host. The retailer may perform this service for the subscriber.

The POD and Host shall support an ID Reporting Screen that displays all of the information required to identify the POD and Host to the service provider. Display of this screen may be initiated by the POD via the MMI or by the Host via the Application_info APDU with application_type 0x01. This screen shall include the following numeric information:

1. POD_ID and Luhn digit shall be displayed combined as 13 decimal digits[1] with sequence[2]:
   M-MMU-UUU-UUU-UUL where the digits are all decimal defined by the fields
   M = PHICA assigned POD Manufacturer number,
   U = Manufacturer assigned Unit ID,
   L = Luhn check digit.

2. Host_ID and Luhn digit shall be displayed combined as 13 decimal digits with sequence:
   M-MMU-UUU-UUU-UUL where the digits are all decimal defined by the fields
   M = PHICA assigned Host Manufacturer number,
   U = Manufacturer assigned Unit ID,
   L = Luhn check digit.

3. A phone number to contact the MSO, defined in the example by the X.

In a system with two-way RF or telco return functionality (Host, Cable plant or phone line, and Headend all support compatible connections) the Host_ID and POD_ID may be sent to the Headend in an authenticated CA System message.

For one-way Cable systems, unidirectional Hosts, or any system without an automatic report-back mechanism, the POD and Host ID's must be reported manually. The POD module shall determine if the Host is unidirectional by sending the oob_tx_tune_req() APDU and receiving the oob_tx_tune_cnf() APDU. If the status_field is a 0x01 (RF transmitter not physically available), then the POD module shall define the Host as unidirectional. The POD module shall also have a means of determining if the system it is resident in is unidirectional.

Following power-up if the POD module determines it has not bound to the Host and is either in a unidirectional system or is inserted into a unidirectional Host, the POD module shall open a session to the Host's MMI resource (if not already open), and send an open MMI dialog request.

If the Host is in an off state or any non-video viewing state, it shall deny the dialog open request. When the Host is in a video viewing state, it shall grant the open MMI dialog request. The POD module shall then send a message containing the ID Reporting Screen. The Host shall display the message and confirm to the POD module that the message has been displayed to the customer.

The ID Reporting Screen shall be displayed only if:

1) the message is selected through a Host user menu system,

---

[1] The PHICA supplied Security Specification details use of POD_ID and Host_ID as binary and decimal values.

[2] Note: The sequence of the digits is normative. The use of dashes is informative.

2) the user selects a program with CP active (EMI ≠ 0) before the Host is validated, or

3) the POD initiates the message display, e.g., at the request of the CA System.

Figure 3.2-A is an example ID Reporting Screen[3]. The POD and Host ID's are each presented as 13 decimal digits including a Luhn check digit.

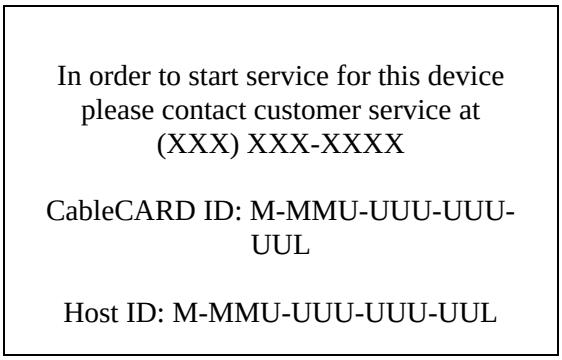

Figure 3.2-A Example ID Reporting Screen

3.2.2 AUTHENTICATION PHASE 1 – CERTIFICATE VERIFICATION & DH KEY EXCHANGE

At the first step of the POD CPS authentication protocol, the Host Certificate List, POD Certificate List, signed data, and Diffie-Hellman public keys are exchanged between the POD and Host. Prior to that, the POD is authorized only for programs with EMI data set to a value of 00 ("copying permitted") if otherwise authorized by the CA system. The first step authentication is achieved based on whether the signatures contained in the Host Certificate List along with the signature on the Diffie-Hellman public key can be verified by the POD and on whether the signatures contained in the POD Certificate List along with the signature on the Diffie-Hellman public key can be verified by the Host.

If the Host certificate list verifies along with the signature on the Diffie-Hellman public key, the Host_ID can then be extracted from the certificate. If the POD certificate list verifies along with the signature on the Diffie-Hellman public key, the POD_ID can then be extracted from that device certificate.

If any part of the copy protection system fails, including certificate verification, the POD shall not perform the CA System decryption step (even if the subscriber would otherwise be authorized to receive the service), the POD module shall then request to open a session to the MMI resource resident on the Host (if not already open), and then send a open MMI dialog request. If the Host is in an off state or any non-video viewing state, it shall deny the dialog open request. When the Host is in a video viewing state, it shall grant the open MMI dialog request. The POD module shall then send a message to the Host similar to that shown below in Figure 3.2-B and shall notify the headend CA System (if possible).

[3] The text portion of this screen is informative. The numeric fields are normative as specified in this section.

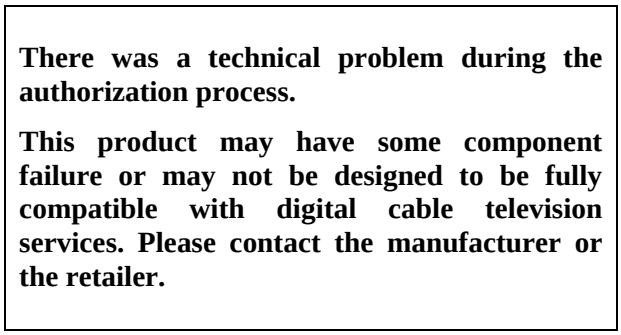

Figure 3.2-B Example CP System Failure Notification message

Thereafter, the failure notification message shall be displayed only if the verification of the Host Certificate or POD Certificate has failed and 1) the message is selected through the menu, or 2) the user tunes to a scrambled channel protected by the CA system.

3.2.3 AUTHENTICATION PHASE 2 – AUTHENTICATION KEY VERIFICATION

A long-term "Authentication Key" ($AuthKey_P$ and $AuthKey_H$) is derived based on the information exchanged between the POD and Host during the first step of authentication. This Authentication Key is calculated as a function of the Host_ID, the POD_ID, and the Diffie Hellman public keys

Both the POD and the Host calculate an AuthKey, as described in the Cryptographic Functions section (section 4). The POD Module sends a request message to the Host to request the Authentication Key derived by the Host. If the POD Module confirms that its derived Authentication Key is the same as the one received from the Host, then the POD accepts that Host as legitimate. This derived Authentication Key shall be stored in non-volatile memory, and can be used later in the calculation of the Copy Protection Key. If a matching AuthKey has not been received within five seconds of the request message, the POD shall not perform the CA system decryption step (even if the subscriber would otherwise be authorized to receive the service) and display the MMI message and report back to the Headend as described above in section 3.2.2.

Authentication at this step is achieved based on the Host being able to prove that it can derive the same shared DH secret key as the POD.

3.2.4 AUTHENTICATION PHASE 3 – HEADEND REPORT BACK

The POD Module will request headend validation checks on the POD and Host ID's. The POD CP validation process requires the CA System to check if the Host_ID and the POD_ID's are listed in the CRLs stored in the headend.

3.2.5 HEADEND REPORT BACK METHODS

3.2.5.1 ONE-WAY SYSTEM DEVICE REGISTRATION AND VALIDATION

After the POD Module verifies the Host certificate list, the Host_ID can be extracted from the Host's device certificate. After the Host Module verifies the POD certificate list, the POD_ID can be extracted from the POD's device certificate. Since there is no upstream connectivity to the headend in a one-way system, the POD Module must rely on the consumer to report this ID information back to the headend, typically by telephone. The POD receives private messages from the headend that are conveyed within the CA System, so these messages are not defined here. The following registration and validation protocol shall be used:

1. The POD stores the Host_ID extracted from the Host device certificate and the POD_ID extracted from the POD device certificate so that they can be compared against the Validated_Host_ID and Validated_POD_ID, received back from the headend in step 6 below.
2. The POD sends a display message request to the Host. This message contains the Host_ID and the POD_ID. The last digit (rightmost) displayed in the Host_ID and POD_ID shall be a single check digit of the ID using the Luhn Check Digit Algorithm described in appendix A.
3. The Host responds with a confirmation message indicating that the message has been displayed to the consumer. The Host also displays the Host_ID and POD_ID to the user in decimal format (digits 0-9).
4. The Host_ID and POD_ID must be reported to the cable operator. One method is by reading the Host_ID and POD_ID to a customer service representative (CSR) over the telephone. Other methods of transferring these data may be used.
5. The CSR records the pairing between the Host_ID and POD_ID, as received from the user. The CSR sends the Host_ID validation request to the CA System. The CA System records the pairing between Host_ID and POD_ID.
6. The CA System holds X.509 Certificate Revocation Lists and checks if the Host_ID and POD_IDs are listed as revoked.
    a) If the Host_ID is not found in the CRL, the CA System re-sends the Host_ID and POD_ID back to the POD Module in a private authenticated CA System Host_ID validation message. Once this message has been sent to the POD, the CA System is allowed to authorize the POD for high value services with EMI values equal to 01, 10, or 11. See section 6, CCI.
    b) This Host_ID validation message may be sent back to the POD substantially later in time than when POD-Host registration begins or the Host_ID is received from the POD. Until the POD receives this message, the POD shall restrict its authorized services only to those services with a EMI value of 00.
    c) If the Host_ID is found in the CRL, the CA System shall mark that Host_ID as fraudulent, and is prohibited from authorizing that Host's associated POD for high value services.
    d) If the POD_ID is found in the CRL, the CA System shall mark that POD_ID as fraudulent, and is prohibited from authorizing that POD for high value services.

7. The CA System sends an EMM to the POD to authorize appropriate services. In the event that some of these services have EMI values of 01, 10, or 11, the POD is prohibited from authorizing[4] those services. The POD shall not authorize services with these EMI values until the Host_ID validation message has been received, even if these services are authorized in an EMM.
8. The POD authenticates the device validation message, to verify that the message did originate in the headend.and then compares the Validated_Host_ID with the Host_ID extracted from the Host device certificate at step 1 above. The Validated_POD_ID is also compared with the POD_ID extracted from the POD device certificate at step 1 above.
9. The POD shall store the validated Host_ID in non-volatile memory so that headend validation need not be conducted every time there is a power down.

Given this protocol, the headend always has an opportunity to revoke the services of a Host using CA System EMMs, and no CRLs need exist in the cable network. Revocation CRLs are only used in the headend in step 6 above. Further, the CA System headend can receive new CRLs, look up new Host_IDs, and deauthorize any compromised Host associated with the POD at any time - all using EMMs. Host selective service revocation issues are discussed more in detail in Section 5.

The POD shall store the Host_ID validated by the headend in non-volatile memory so that the headend's validation need not be conducted every time there is a power down.

3.2.5.2 MANUAL RETURN AUTHENTICATION – ERROR AND OTHER CONDITIONS

The subscriber may request the ID Reporting Screen, typically from a Host menu. There are two conditions in which the ID Reporting Screen is not valid:

- The Host and POD module have not completed the binding process.
- The Host has an invalid certificate.

It should be noted that the following messages are displayed when diagnostic messages are requested and will be informational as opposed to friendly user interfaces. These messages are suggestions only and may be modified to an equivalent message.

Incomplete Binding

In the event that the Host and POD module have not completed phase one of binding (validation of the Host certificate) at the time the Host requests the copy protection message screen, the POD module shall display a message indicating it is not available, for example:

Information not Available

[4] The POD authorizes the Host by decrypting CA-encrypted services and re-encrypting them for the Host. When "the POD shall not authorize a service for the Host" the POD shall not perform the CA-decryption step (even if the subscriber would otherwise be authorized to receive the service).

Invalid Certificate

In the event that the POD supplies an invalid certificate to the Host the Host shall display a message informing the user, for example:

POD Certificate Invalid

In the event that the Host supplies an invalid certificate to the POD module, the POD shall request the copy protection message screen and display a message informing the user, for example:

Host Certificate Invalid

3.2.5.3 TWO-WAY SYSTEM POD AND HOST ID VALIDATION

The POD Module in a two-way system has upstream connectivity to the headend. Therefore, the POD Module can send the POD_ID and Host_ID directly to the headend in an authenticated manner without requiring a consumer to read these IDs back to a CSR over the telephone. The Host_ID is extracted from the Host certificate after the Host certificate is verified by the POD Module during the first step of authentication. In the two-way system, the following registration and validation protocol shall be used:

1. The POD stores the Host_ID extracted from the Host device certificate so it can be compared against the Host_ID received back from the headend in step 4 below.
2. The POD stores the POD_ID extracted from the POD device certificate so it can be compared against the POD_ID received back from the headend in step 4 below.
3. The POD sends the Host_ID and POD_ID through the upstream OOB or Host modem resource in a private authenticated CA System message. The headend CA System records the pairing between Host_ID and POD_ID.
4. The CA System holds X.509 Certificate Revocation Lists (CRLs) and checks if Host_ID and the POD_ID are listed as revoked.
    a) If the Host_ID and the POD_IDs are <u>not found</u> in the CRL, the CA System re-sends the Host_ID and POD_ID back to the POD Module in a private authenticated CA System Host_ID validation message. Once this message has been sent to the POD, the CA System is allowed to authorize the POD for high value services with EMI values equal to 01, 10, or 11. See section 6, CCI.
    b) This Host_ID validation message may be sent back to the POD substantially later in time than when POD-Host registration begins or the Host_ID is received from the POD. Until the POD receives this message, the POD shall restrict its authorized services only to those services with a EMI value of 00.

c) If the Host_ID or POD_ID are found in the CRL, the CA System shall mark that Host_ID or POD_ID as fraudulent, and is prohibited from authorizing that Host's associated POD for high value services

5. The CA System sends an EMM to the POD to authorize appropriate services. In the event that some of these services have EMI values of 01, 10, or 11, the POD is prohibited from authorizing those services. The POD shall not authorize services with these EMI values until the Host_ID validation message has been received, even if these services are authorized in an EMM.
6. The POD authenticates the Host_ID validation message and then compares the Host_ID with the Host_ID extracted from the Host device certificate at step 1 above. The authentication is to verify that the message did originate in the headend.
7. The POD also then compares the POD_ID with the POD_ID extracted from the POD device certificate at step 2 above. The authentication is to verify that the message did originate in the headend.
8. The POD shall store the Host_ID and POD_ID, validated by the headend, in non-volatile memory so that headend validation need not be conducted every time there is a power down.

Given this protocol, the headend always has an opportunity to revoke using CA System EMMs, and no CRLs need exist in the cable network. CRLs are used in the headend only, in step 4. Further, the CA System headend can receive new CRLs, look up new Host_IDs and POD_IDs, and then revoke selected services of any compromised Hosts associated with CA System PODs at any time - all using CA System EMMs. Host selective service revocation issues are discussed in more detail in a later section.

3.3 POWER-UP RE-AUTHENTICATION

At power-up, if the POD detects that it holds an Authentication Key from a previous binding in non-volatile memory, the POD shall attempt a re-authentication procedure. This procedure will determine if this is the Host to which the POD was last bound and if the POD is the same to which the Host was last bound. The POD initiates the re-authentication by requesting that the Host send its $AuthKey_H$. If the received $AuthKey_H$ does not match the POD's stored $AuthKey_P$, the POD shall reject the Host and re-initialize the binding procedure between the POD and Host as described in section 3.2. If the authentication keys do match, then the certification verification is considered valid, the private DH Key and authentication keys are preserved, and only the Copy Protection Key is regenerated.

3.4 POD OPERATION WITH MULTIPLE HOSTS

Each POD shall bind to exactly one Host at a time. No POD shall store two or more sets of Authentication Keys or other Host-specific information. A given POD can be removed from a Host and inserted into a different Host at any time. The re-authentication procedure will indicate a mismatch in authentication keys, and the POD shall initiate the binding procedure, including full Host Certificate verification. If this POD is later returned to the previous Host it shall again initiate the binding procedure, as it has authentication information only on the last Host to which it was bound.

3.5 HOST OPERATION WITH MULTIPLE PODS

Host operation with multiple POD modules is beyond the scope of this document and is subject to further study.

4. CRYPTOGRAPHIC FUNCTIONS

The basic key negotiation process for POD copy protection is shown in Figure 4.0-A below.

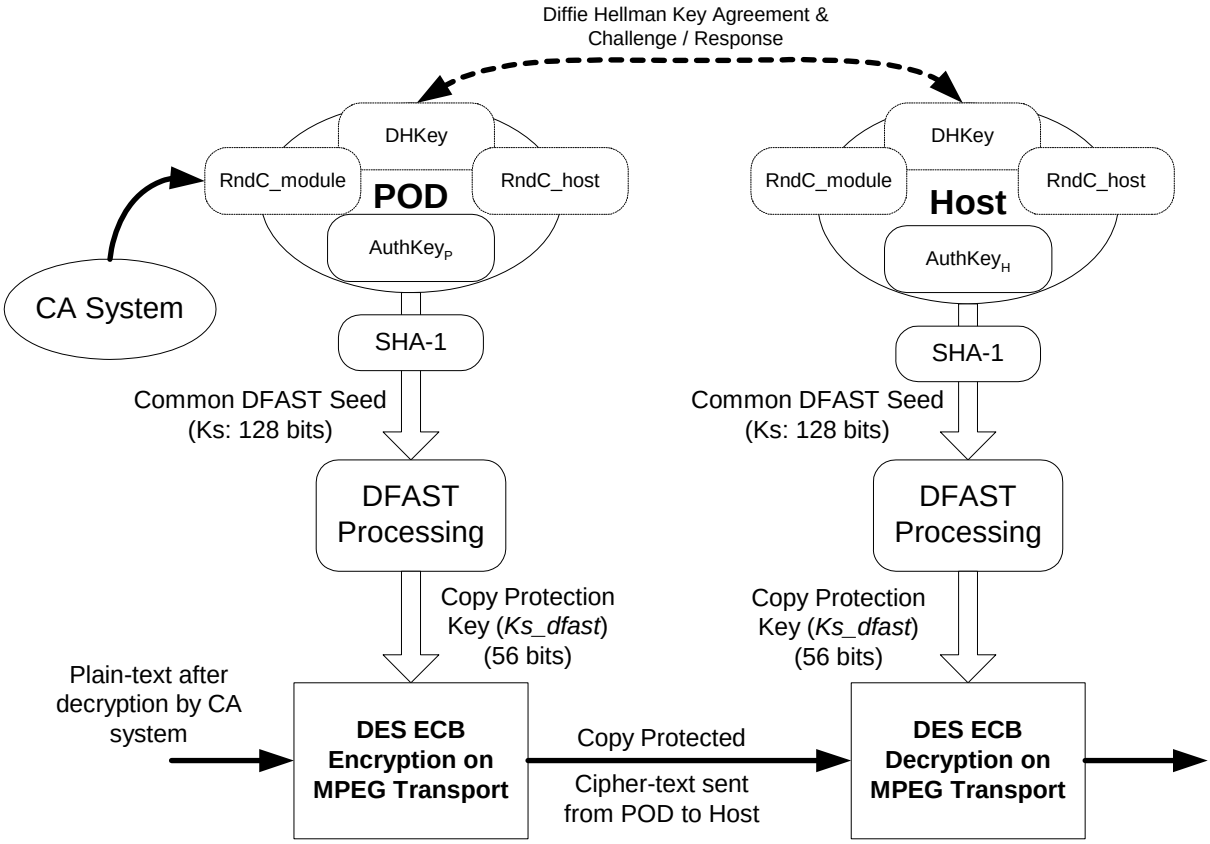

Figure 4.0-A Overall POD Copy Protection

4.1 AUTHENTICATION KEY GENERATION

During the Host authentication process, Diffie-Hellman public keys are exchanged between POD and Host as a conventional part of the DH protocol. DH public keys along with the IDs are used to derive the authentication keys which authenticate the DH exchange and resist "Man in the Middle" attacks. Both POD and Host calculate a 160 bit value called the Authentication Key or AuthKey. The AuthKey is calculated based on the 64 bit POD Module ID, and the 40 bit Host_ID, and the shared Diffie Hellman Secret Key (DHKey). The Host transfers its calculation of this value to the POD, where the POD compares it against the one it calculated from the same information. The POD proceeds with authentication if and only if its internally calculated version of the AuthKey is identical to the one it obtains from the Host.

The Diffie-Hellman shared secret key (DHKey) is computed as:

$$DHKey_P = (DH\_pubKey_H)^x \bmod n = (DH\_pubKey_P)^y \bmod n = DHKey_H$$

The POD computes its authentication key by applying the SHA-1 function:

$AuthKey_P$ = SHA-1 [*DHKey* | Host_ID | POD_ID]

The Host also computes its authentication key by applying the SHA-1 function:

$AuthKey_H$ = SHA-1 [*DHKey* | Host_ID | POD_ID]

$AuthKey_P$ and $AuthKey_H$ are used in Copy Protection Key Generation, as described in section 4.2 below.

Authentication Key generation need occur only once (per Host-POD pair) when the POD and Host are first connected. The resulting $AuthKey_P$ and $AuthKey_H$ values and Diffie-Hellman Secret Key (DHKey) then need to be stored in non-volatile memory, and are used to generate transmission keys later in the key derivation step.

4.2 COPY PROTECTION KEY GENERATION

A series of steps is employed to generate and refresh the CP-Key (Ks_dfast) at the following times:

- At the end of the authentication process;
- Periodically at a rate set by max_key_session_period;
- At every power cycle;
- When initiated by the CA System; and
- At every hard reset.

Highly randomized variables are used as new random numbers ("nonces"). Random nonces along with IDs are exchanged between the POD and Host interface. A common Copy Protection Key between the POD and Host is derived from these newly exchanged random numbers, the Authentication Key ($AuthKey_P$ *or* $AuthKey_H$) and the 1024 bit shared secret Diffie-Hellman key (*DHKey*). The derived common Copy Protection Key (*Ks_dfast*) is then used to encrypt/decrypt MPEG data sent from the POD to the Host.

4.2.1 BASIC KEY GENERATION PROTOCOL

The following procedure shall be followed to generate the CP-Key:

1) POD checks whether a previously derived authentication key is already stored in dedicated non-volatile memory. If such an $AuthKey_P$ is present, then continue to the next step. Otherwise, restart the whole authentication process as detailed in Section 3.2.

2) The POD checks if the received validated ID's are equal to the previously stored ID. If they are the same, the POD shall proceed with the key generation process; otherwise, the POD shall only authorize services with EMI values of 00.

3) The POD generates its 64 bit random number (*N_module*) .

4) The POD sends this *N_module* and its ID (*POD_ID*) in the clear to the Host.

5) The Host generates its 64 bits random number (*N_Host*).

6) The Host sends N_Host and its Host_ID in the clear to the POD.

7) The POD computes the Copy Protection Key based on long-term keys and newly exchanged random number using the SHA-1 hash function and the DFAST algorithm, as described in the following section.

8) The Host computes the Copy Protection Key also based on long-term keys and newly exchanged random number using the SHA-1 hash function and the DFAST algorithm, as described in the following section.

4.2.2 POD MODULE COPY PROTECTION KEY

The SHA-1 function is first used to hash the long-term keys, $AuthKey_P$ and the *DHKey*, and the random numbers exchanged for key generation. The result is named *Ks*:

$$Ks = \text{SHA-1}\ [AuthKey_P \mid DHKey \mid N\_Host \mid N\_module]_{\text{MSB128}}$$

The 160-bit SHA-1 output is truncated to its 128 MSB's, left-most bits, to generate a seed, Ks, with the proper 128 bit length for the input to the DFAST engine. The DFAST algorithm is applied to *Ks* to produce the 56-bit value of the Copy Protection Key, also known as *Ks_dfast*:

$$\text{CP-Key} = Ks\_dfast = \text{DFAST}\ [\ Ks\ ]$$

DFAST details are specified in a separate document; contact the PHICA. Table 4.2-A defines the size of keys, as well as the parameters used to derive them.

Table 4.2-A Length of Keys and Parameters Used in the Key Generation

Key or Variable	Size (bits)	Description
Nonces (*N_Host, N_module)*	64 bits each	Random numbers used to refresh the CP-Key.
Authentication Keys ($AuthKey_H$, $AuthKey_P$)	160 bits each	Results from the Host authentication process. It is a long-term key, and is stored in a non-volatile memory.
Shared Diffie-Hellman Key (*DHKey*)	1024 bits	The 1024 bit shared DH secret key. It is a long-term key, and is stored in non-volatile memory.
SHA-1 Key (*Ks)*	128 bits	The most significant 128 bits of the 160 bit SHA-1 output, where the SHA-1 input is the DHKey, Authentication Key, and nonces from POD and Host.
Copy Protection Key (*Ks_dfast)*	56 bits	DFAST output, final encryption and decryption key

4.2.3 HOST COPY PROTECTION KEY

The Host computes its SHA-1 key, Ks, based on the Authentication Key ($AuthKey_H$), shared secret DH key (*DHKey*), and the random numbers exchanged in the key generation:

$$Ks = \text{SHA-1}\,[AuthKey_H \mid DHKey \mid N\_Host \mid N\_module]_{\,\text{MSB128}}$$

The 160-bit SHA-1 output is truncated to its 128 MSB's, left-most bits, to generate a seed, Ks, with the proper length for the DFAST engine. The DFAST algorithm is applied to *Ks* to produce the 56-bit value of the Copy Protection Key:

$$\text{CP-Key} = Ks\_dfast = \text{DFAST}\,[\,Ks\,]$$

4.3 CP-KEY REFRESH

The CP-Key shall refresh periodically as initiated by the POD. The CA System will set the refresh period with a parameter, max_key_session_period, transmitted to the POD by the CA System with maximum security.

For each single CP-Key refresh: after the POD initiates a CP-Key refresh cycle it shall start a Key Refresh timer. The POD shall stop scrambling the selected program during the synchronization of keys. It shall start to encrypt again on the earlier of; successful completion of the authenticated CP-Key refresh cycle, or transmitting unencrypted data for one second. The CCI shall not be changed during this <1 second period.

Each CP-Key refresh shall recalculate the content key using a new pair of nonces (N_Host, N_module) exchanged between the POD and Host.

Note that the POD requests the Host's Authentication Key at every power up or hard reset if it has a valid Authentication Key stored in non-volatile memory. The POD compares the received *AuthKeyH* to its stored *AuthKeyP* to detect if it has been inserted into a new Host or if the Host has been bound to a different POD. If the authentication keys match, then the POD shall initiate a CP-Key refresh. (If a valid $AuthKey_P$ is not found, the POD initiates a full binding process.)

4.3.1 KEY SESSION PERIOD

The key session period is the period of time in which the POD module and Host utilize the same key for copy protection. There is a maximum length of this period, **max_key_session_period**, programmable by the CA System. The POD module shall implement a timer which is not dependent on the program selected by the Host, and is reset anytime new keys are exchanged between the Host and the POD module. If this timer reaches the value of the **max_key_session_period**, the POD module shall initiate a CP-Key refresh. The max_key_session_period shall be implemented as a 16 bit value with a resolution of 10 seconds (one decasecond). If the value of max_key_session_period is zero, then the maximum key session period is unlimited. The Host is not aware of max_key_session_period.

4.3.2 KEY REFRESH PERIOD

The POD controls the timing of the key refresh cycle. When the POD sends its nonce to the Host in the CP_data_req() message, the POD starts a Key Refresh Timer. When the Host receives the CP_data_req(), the Host generates its nonce and sends it to the POD in the CP_data_cnf() message. The Host shall be implemented such that it shall transmit a CP_data_cnf() message within one second of receiving a CP_data_req() message.

The POD and Host shall start the calculation of the Copy Protection keys when that Host issues the CP_data_cnf(). The POD and Host shall both be implemented such that each calculates its Copy Protection key within eight seconds. The POD shall send the CP_sync_req() to the Host when the Key Refresh Timer reaches nine seconds. This timing ensures that both the POD and Host have a minimum of eight seconds to complete key calculation. The POD CP_sync_req() message indicates that the POD has completed calculation of the Copy Protection key . The Host shall issue the CP_sync_cnf() message when it has received the CP_sync_req() message and has completed calculation of the Host Copy Protection key.

In single CP-Key operation, when the POD issues the CP_sync_req() message, the POD module shall turn off scrambling of the MPEG content output and set the MPEG transport_scrambling_control to 00. The Host receives cleartext packets and will recognize these packets as unencrypted according to MPEG rules. When the Key Refresh Timer reaches ten seconds, the POD shall immediately return to scrambling of MPEG packets. If the key refresh has not completed when the Key Refresh Timer reaches ten seconds, the POD module shall disable CA decryption of copy protected content until a full CP-Key refresh is completed.

In dual CP-Key operation all protected content shall be scrambled throughout the CP-Key generation and change process. No one-second clear period shall occur. This is the primary objective of the dual key capability.

Figure 4.3-A below defines the POD flow during a Key Refresh cycle. Figure 4.3-B below defines the Host flow during a Key Refresh cycles.

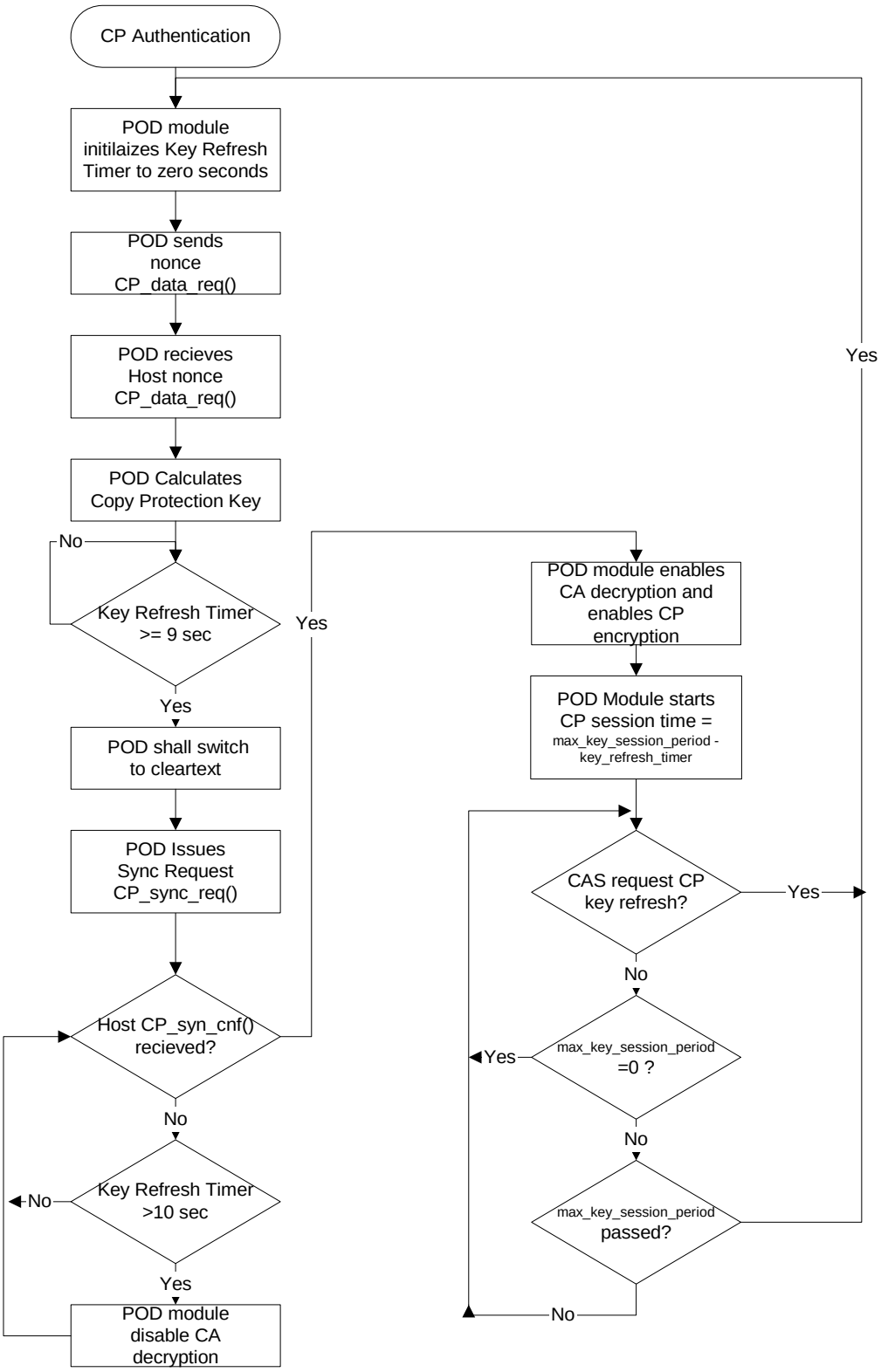

Figure 4.3-A POD CP-Key Refresh Session Flow Chart

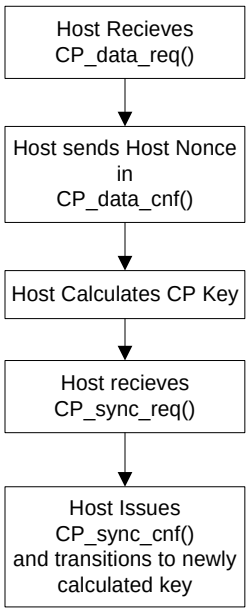

Figure 4.3-B. Host CP-Key Refresh Flow Chart

4.3.3 CA SYSTEM KEY REFRESH

The POD module shall be capable of initiating a key refresh at the command of the CA System. This key refresh command shall occur regardless of any other conditions, excepting that a key refresh is occurring at that time.

4.3.4 KEY REFRESH INITIALIZATION

After the copy protection authorization process has occurred, an initial key refresh shall occur.

4.3.5 CHANNEL CHANGE

When a channel change occurs, the Host shall treat all CP-scrambled content as if the EMI is set to "copy never". The Host shall immediately begin using the value of EMI when it is received from the POD. Channel change shall not cause a key refresh to occur.

4.3.6 TWO KEY SYNCHRONIZATION MODE (INFORMATIVE)

Optionally, a POD or Host may implement key refresh using a system of EVEN and ODD CP-Key's. In that case the above described system of going into the clear for one second is not needed, and is instead replaced with a one second period before a new key is written into one of the EVEN or ODD key registers. Such a two key system shall not be fully specified at this time, but shall be fully specified in a future release.

The presence of two CP-Key registers and selection logic for EVEN and ODD CP-Key's based on MPEG-TS header transport_scrambling_control bits, shall be optional in both POD and Host. If implemented the transport_scrambling_control bits shall select the EVEN or ODD key.

It is highly recommended that POD and Host manufacturers build silicon that is capable of holding both an EVEN and an ODD CP-Key, and is capable of properly selecting the correct EVEN or ODD key based on the transport_scrambling_control bits. It is further recommended that all POD and Host devices include a firmware download means to fully support ODD/EVEN CP-Key refresh when it is fully defined, e.g., for APDUs, protocols flows, etc.

4.3.7 TRANSPORT SCRAMBLING CONTROL FIELD

The transport_scrambling_control field of the MPEG transport stream packet provides control information for key changes. The transport_scrambling_control field bit values for single CP-Key and dual CP-Key modes shall be defined as in Table 4.3-A.

Table 4.3-A MPEG Transport_scrambling_control values

Bit Values	For Single CP-Key Mode	For Optional Dual CP-Key Mode
00	No scrambling of TS packet payload	No scrambling of TS packet payload
01	Reserved	Reserved
10	Reserved	TS packet scrambled using EVEN key
11	Transport packet scrambled	TS packet scrambled using ODD key

4.4 DIFFIE-HELLMAN KEY EXCHANGE ALGORITHM

4.4.1 ALGORITHM OVERVIEW

Diffie-Hellman Public Key Agreement algorithm provides a method for POD and Host to compute a long term shared secret that is used in the encryption/decryption key generation. The Diffie-Hellman protocol provides the system with a cryptographic property known as “perfect forward secrecy”. Figure 4.4-A illustrates the two-step Diffie-Hellman operations conducted in the POD, Host and interface between them.

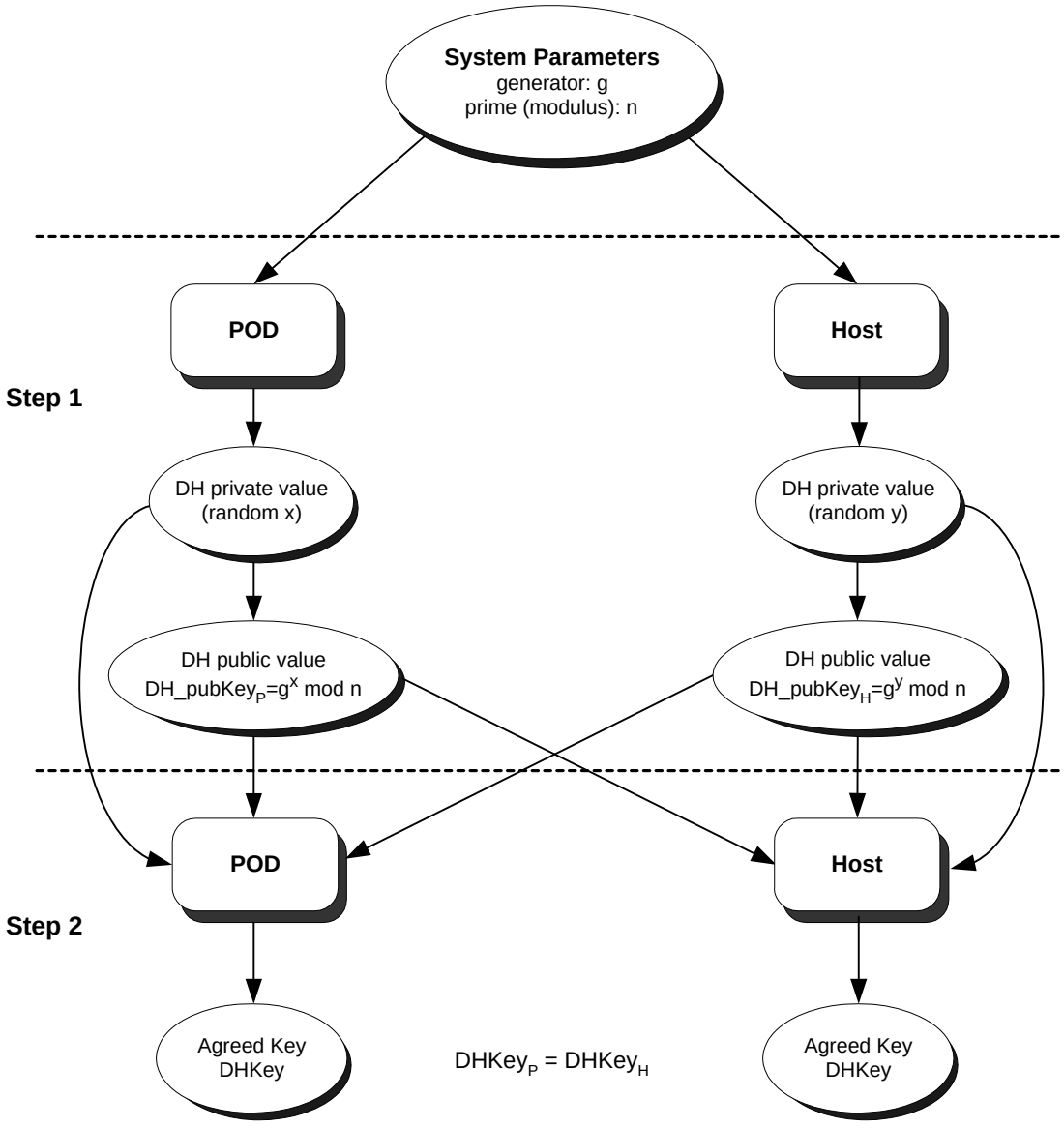

Figure 4.4-A Diffie-Hellman Key Agreement Protocol between POD and Host

4.4.2 ALGORITHM IMPLEMENTATION

4.4.2.1 SYSTEM PARAMETER GENERATION [INFORMATIVE]

The length of the modulus (n) is usually chosen to have a comparable level of difficulty against the best discrete logarithm algorithm. A 1024-bit prime (modulus) is currently considered sufficient against attack. The length of generator is the same as the length of modulus.

These constants are provided by the PHICA with the POD-Host Interface License.

4.4.2.2 STEP 1 OPERATIONS

The POD and Host each execute the Diffie-Hellman protocol as follows:

1. The POD randomly generates a private exponent, x, where $1 \leq x \leq n - 2$, where the exponent x need not have the full 1024 bit length. The exponent length shall be at least 160 bits long.
2. The Host randomly generates its private exponent, y, where $1 \leq y \leq n - 2$, selecting y to have the length of at least 160 bits.
3. The POD computes its public key value $DH\_pubKey_P = g^x \bmod n$, and sends it to the Host along with its POD_ID.
4. The Host computes its public key value $DH\_pubKey_H = g^y \bmod n$, and sends it to POD along with its Host Certificate.
5. The DH public keys are generated in such a way that computing the private exponent from the public value is computationally infeasible.

4.4.2.3 STEP 2 OPERATIONS

Both POD and Host compute the agreed-upon secret key using the other's public value, their own private value, and the system parameters modulus n, as follows:

1. The POD derives the 1024 bit shared key $DHKey_P = (DH\_pubKey_H)^x \bmod n$; and
2. The Host derives the 1024 bit shared key $DHKey_H = (DH\_pubKey_P)^y \bmod n$;

Even though both the POD and Host are making computations using different private values (x, y), they end up with the same secret DHKey where:

$$DHKey_P = (g^y)^x \bmod n = g^{yx} \bmod n = (g^x)^y \bmod n = g^{xy} \bmod n = DHKey_H$$

4.4.2.4 DHKEY EXCHANGE AND HOST AUTHENTICATION [INFORMATIVE]

Note that Step 1 operations are performed in Host Authentication Phase 1 described in section 3.2.2. Also note that the product of Step 2 Operations, the shared DHKeys, are use for the CP-Key generation. CP-Key generation is initiated by the POD only after all three phases of the Host Authentication have been completed successfully.

4.4.2.5 REPRESENTATION OF LARGE VALUES AS OCTETS [INFORMATIVE]

To represent large parameter values, like the 1024-bit modulus, as a series of octets (bytes) the most significant bit (MSB) of the first octet should represent the MSB of the value, the least significant bit (LSB) of the first octet the eighth MSB of the value, continuing until the LSB of the value becomes the LSB of the last octet. In other words, the first octet in the series has the most significance in the integer and the last octet has the least significance.

A large parameter z of length k*8 bits should be converted into an octet block PV of length k such that:

$$z = \sum_{i=1}^{k} 2^{8(k-i)} PV_i$$

where PV_1, ..., PV_k are the octets of PV from first to last.

4.5 SHA-1 SECURE HASH ALGORITHM

The POD Copy Protection specification employs the RSA signature algorithm with SHA-1 for all X.509 digital certificates. The POD Copy Protection specification uses the value F_4 (65537 decimal, 010001 Hex) as the public exponent for its signing operation. The Device Root PHICA will employ a modulus length of 2048 bits for signing the Manufacturer XCA certificates it issues. Manufacturer XCA's shall employ signature key modulus length of 2048 bits.

The following functions and operations use the SHA-1 algorithm:

- Host Certificate Signature Verification: the signature algorithm is based on the RSA digital signature scheme defined in FIPS 180-1, which uses the SHA-1 primitive.
- POD Certificate Signature Verification: the signature algorithm is based on the RSA digital signature scheme defined in FIPS 180-1, which uses the SHA-1 primitive.
- Authentication key generation as described in section 4.1 above.
- Copy Protection Key generation, as described in section 4.2.

4.6 RANDOM NUMBER GENERATION

If a pseudorandom integer generator is used to generate *N_Host* and *N_module* as well as DH private keys (x and y), it shall be compliant with the SHA-1 based algorithm described in FIPS PUB 186-1, Appendix 3, Section 3.3. The POD and Host shall each have a uniquely generated seed value that is set in the factory. A physical random number generator may be implemented as the seed generator. The seed generator shall comply with the FIPS PUB 140-1 Section 4.11.1 test for randomness.

4.7 DFAST ALGORITHM

4.7.1 ALGORITHM OVERVIEW

The diagram in Figure 4.0-A at the beginning of this chapter shows how DFAST would be used. Detailed information on DFAST design and implementation is presented in the document "DFAST Implementation Description" obtained from the PHICA.

4.7.2 DFAST CHARACTERISTICS

Accepts a 128 bit input value (*Ks*) and generates 56 bits of output (*Ks_dfast*, the Copy Protection Key). This output from the DFAST function is used as the DES ECB key for copy protection content scrambling and descrambling;

4.8 RSA DIGITAL SIGNATURES

RSA digital signatures shall be computed using block type 01 as specified in PKCS #1 version 1.5, normative reference 9, in section 1.2.1 above.

5. HOST SERVICE REVOCATION MECHANISMS

5.1 SYSTEM ISSUES

This section addresses details of revocation of selected Host services including CRL-based Host revocation mechanisms, fundamental principles, and the circumstances under which revocation should occur.

5.2 REVOCATION CIRCUMSTANCES [INFORMATIVE]

The revocation technique is used as a safeguard to prevent copy protected content from delivery to insecure Hosts. Cases where this might occur include:

1. A fraudulent Host claimed to be compliant with obligations contained in the DFAST license;
2. An erroneously-designed Host claimed to be compliant with obligations contained in the DFAST license;
3. Implementations that were compliant at the time they were distributed become non-compliant because of a change in circumstances (e.g. wide consumer availability of de-bugging programs or a specific security scheme that has been compromised).

5.3 FRAUDULENT HOST IDENTIFICATION

This POD-CP system requires the POD Module to retrieve the Host Certificate List from the Host and validate this Host Certificate List. The POD-CP system also requires that the Host retrieve the POD Certificate List from the POD and validate it. The POD then reports this Host_ID and the POD_ID to the headend when a POD Module is bound with a Host and two-way capabilities are available. When two-way capability is not available, the Host_ID and POD_ID are reported to the headend by the end-user typically via telephone (touch-tone keypad or voice).

In all cases the POD and Host ID's shall be reported back to the headend by some means. Except for the case of a copy protection system failure, an invalid POD Certificate Data (*POD_DevCert* and *POD_ManCert*) or an invalid Host Certificate Data (*Host_DevCert* and *Host_ManCert*) described in section 3.2.2 and the failure to confirm the Authentication Key described in section 3.3.4, the criterion for identification and resolution of fraudulent Hosts by the operator are not part of this specification. CRL based revocation is performed at the headend not locally in the POD or Host.

5.4 CA SYSTEM REVOCATION & SELECTIVE DENIAL OF SERVICES

5.4.1 DEFINITION OF REVOCATION

Revocation is the concept of denying selected services to invalid or suspect Host devices on the network. This is achieved by not CA authorizing the POD for any high value content for which the bound Host is denied reception.

Knowing the pairing of the POD to the Host is central to the use of CA System revocation. In both one-way and two-way systems the CA System Headend can reliably determine the Host_ID associated with each POD, and can therefore revoke selected services of any Host.

5.4.2 SELECTIVE SERVICE DENIAL

Fundamentally, services rather than Hosts are revoked. A fraudulent Host can still legally watch clear services, for example, as well as free special offerings. *Revocation can be described as a means to automatically control whether high value services are authorized in the specific POD connected to a specific Host.*

A revocation authority and CRL-based service revocation mechanism shall be integrated into the CA System Headend to revoke selected services of a fraudulent or cloned Host. When a Host is found on a CRL in the Headend, the POD associated with that Host shall have its authorized services limited to exclude high value content. High value content is determined by the Copy Control Information defined for that content; see Table 6.1-B. "High value" content is defined as content with EMI values of 01 ("No further copying is permitted"), 10 ("One generation copy is permitted"), or 11 ("Copying is prohibited"). Content marked with EMI value 00 ("Copying not restricted") is not "high value".

The trustworthiness of a Host can be in question due to a number of conditions:

Condition 1. Host authentication protocol failure during POD verification of the Host Certificate.

Condition 2. Host authentication protocol failure to match the Host and POD generated Authentication Keys.

While any of the conditions above persist, the POD shall operate only in pass-through or clear mode.

Condition 3. The Headend may not have confirmed that the Host is valid (not listed on any revocation list). This may be because of real-time processing bottlenecks in the Headend due to either staffing issues or upstream channel congestion.

During this condition the POD shall enable CA-descrambling of content with EMI=0 only. High value services with EMI equal to 01, 10, or 11 shall not be CA-descrambled until the Host validation and binding process is complete.

5.5 THE REVOCATION PROCESS

The administrative process for determining whether or not to revoke selected services of a Host involves a review process through the POD-CP Certificate Authority.

5.6 IMPLEMENTATION IN THE HEADEND

Host selected service revocation is achieved in the POD CPS by processing a certificate revocation message at the headend. Given the protocols defined in section 6.3 and 6.4, the headend always has the opportunity to revoke selected services of a compromised Host or POD using CA System EMMs. Further, the CA System headend can receive new CRLs, look up new Host_IDs and POD_IDs, and then deauthorize specific services granted to the POD associated with any bad Hosts or PODs at any time.

Table 5.6-A CRL Based Host and POD Service Revocation

One-way System (Voice)	Two-way System	Description
RSA based Certificate Verification Voice IDs report-back mechanism IDs vs. CRLs check and return-back by the headend at any time Revocation display message to user	RSA based Certificate Verification IDs report-back mechanism IDs return-back by the headend at any time Revocation display message to user	Compromised Host is listed in CRLs provided by CableLabs, MSO's, or service providers. CRL based revocation is built in the CA System in the headend. If IDs are listed in CRL, a new EMM is triggered into the system to de-authorize the POD - preventing services from being sent to a compromised Host.

6. COPY CONTROL INFORMATION (CCI)

The content provider and the content distributor determine CCI value for each program. The CA System delivers the CCI securely to the POD module. The POD passes CCI to the Host through a secure authentication protocol. The Host uses the CCI to control copy creation, analog output copy control encoding, and to set copy control parameters on Host outputs.

6.1 CCI DEFINITION

CCI is a single byte, 8 bit, field conveyed from POD to Host. Four of the eight bits are defined. The remaining four are reserved. The reserved bits shall be set to zero by the POD as shown in Table 6.1-A. The Host shall use the reserved bit values received from the POD only for execution of the Authenticated Tunnel Protocol described below. The Host shall ignore the reserved bit values thereafter.

Table 6.1-A CCI Bit Assignments

CCI Bits #	7	6	5	4	3	2	1	0
POD sets to	0	0	0	0	APS1	APS0	EMI1	EMI0
Host interprets as	rsvd	rsvd	rsvd	rsvd	APS1	APS0	EMI1	EMI0

6.1.1 EMI - DIGITAL COPY CONTROL BITS

The two LSB's of the CCI byte are the EMI bits. They shall control copy permissions for digital copies. The EMI bits shall be supplied to any Host digital output ports for control of copies made from those outputs. The EMI bits are defined in Table 6.1-B.

Table 6.1-B EMI Values and Content

EMI Value	Digital Copy Permission	Content Type
00	Copying not restricted	Not "High Value"
01	No further copying is permitted.	High Value
10	One generation copy is permitted.	High Value
11	Copying is prohibited.	High Value

6.1.2 APS - ANALOG PROTECTION SYSTEM

Bits 3 and 2 of CCI as shown in Table 6.1-A are the APS bits 1 and 0 respectively. The Host shall use the APS bits to control copy protection encoding of analog composite outputs as described in Table 6.1-C.

Table 6.1-C APS Value Definitions

APS	Description
00	Copy Protection Encoding Off
01	AGC Process On, Split Burst Off
10	AGC Process On, 2 Line Split Burst On
11	AGC Process On, 4 Line Split Burst On

6.2 ASSOCIATING CCI WITH A SERVICE

The CA System shall securely associate CCI with a specific MPEG Program. The MPEG Program Number zero shall not be used for programs covered by this specification.

6.3 CONVEYING CCI FROM HEADEND TO POD

The CA System shall provide a private secure delivery means (e.g. an ECM) to transfer CCI from the Headend to the POD. This delivery means shall preserve the association between CCI and MPEG Program Number.

6.4 CONVEYING CCI FROM POD TO HOST

Delivery of CCI from POD to Host shall be authenticated via the exchange of messages as shown in Figure 6.4-A. The messages are based on a SHA function performed on the CCI, CP-Key and MPEG Program Number.

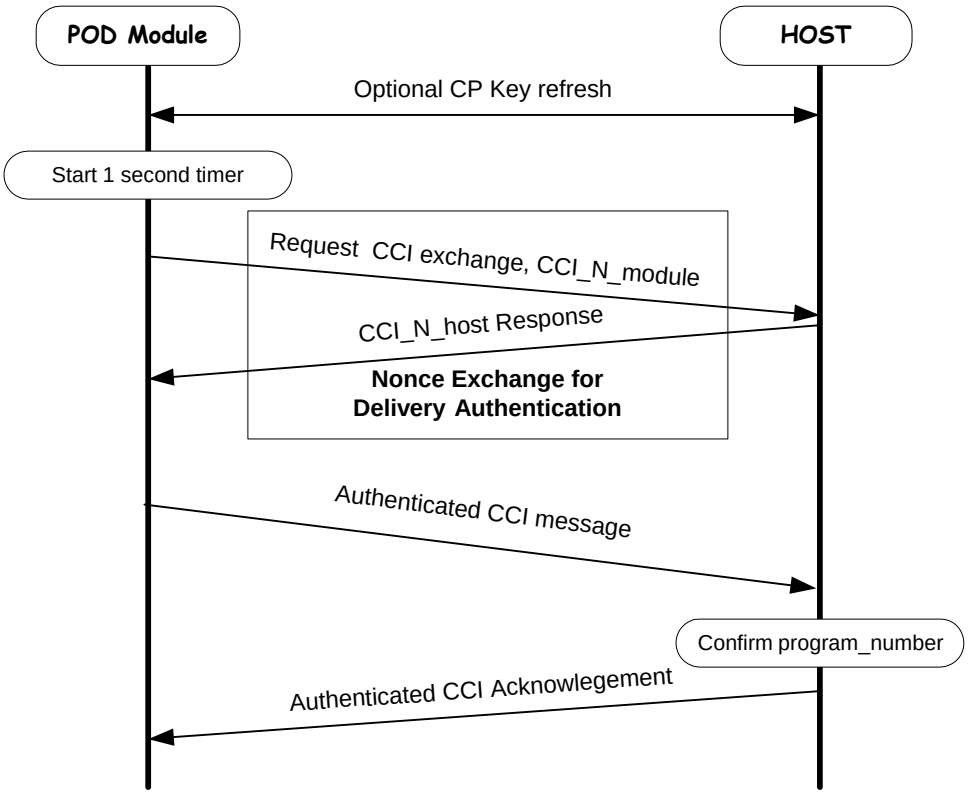

Figure 6.4-A CCI Delivery Sequence

6.4.1 CCI DELIVERY INSTANCES

The POD shall send CCI to the Host only after the POD and Host have successfully bound and negotiated a shared CP-Key. The POD shall initiate CCI transfer to the Host immediately after:

1. the POD tunes to a new MPEG Program by request of the Host, or
2. the MPEG Program Number changes on a tuned ‘channel’, or
3. any change in the CCI bits during a program, or
4. any change in the MPEG packet ID values that the POD is descrambling.

6.4.2 AUTHENTICATED TUNNEL PROTOCOL

The "authenticated tunnel protocol" is a means of verifying delivery of valid CCI from POD to Host. The POD and Host shall jointly execute the steps below once for each transfer of CCI. Any failure of the steps described below shall result in a failed CCI delivery. If the steps above are not completed before the one-second time-out expires the POD shall disable CA-descrambling of copy protected content and the Host shall default to maximal protection of all CP-scrambled content until the CCI delivery protocol completes successfully.

Step 1. The POD generates a new random number CCI_N_module and starts a 1-second time-out.

Step 2. The POD sends CCI_N_module, program_number, and a request for CCI_N_Host.

Step 3. The Host generates a new random number CCI_N_Host.

Step 4. The Host replies with CCI_N_Host and program_number (received in step 2 above).

Step 5. The POD calculates two values: CCI_auth to authenticate CCI delivery, and CCI_ack to authenticate Host acknowledgment of receipt, as:

CCI_auth = SHA-1(CCI | CP-Key | CCI_N_module | CCI_N_Host | program_number)

CCI_ack = SHA-1(CCI | CP-Key | CCI_N_module | CCI_N_Host)

Step 6. The POD transmits CCI_auth, CCI, and program_number to the Host.

Step 7. The Host calculates CCI_auth using the received CCI value and compares it with the CCI_auth value received from the POD. Failed equivalence generates an error condition and the Host sets EMI to 11.

Step 8. The Host shall begin controlling it’s outputs based on valid CCI within one second.

Step 9. The Host calculates CCI_ack and sends it to the POD.

Step 10. The POD compares the received CCI_ack with the value calculated in step 5 above. Failed equivalence generates an error condition.

7. TRANSPORT SCRAMBLING POD TO HOST

MPEG content delivered to the POD module by the Cable network with EMI greater than zero shall be CP-scrambled. MPEG content which is delivered with EMI equal to zero, no copying restrictions specified, for example free access off-air broadcast content, shall be delivered CP-unscrambled from the POD to the Host. Such content may or may not be CA-scrambled during delivery from the Headend to the POD.

7.1 MPEG SCRAMBLING

The POD processes content flowing from input to output in one of four modes:

- Clear: no change of CA-unscrambled and EMI=00 content which remains 'in-the-clear'
- CA-only: descrambles CA-scrambled content marked EMI=00 for output 'in-the-clear'
- Rescramble: CA-descrambles and CP-scrambles content marked EMI>0
- Pass-through: no change of CA-scrambled content (leaving it useless to the Host)

The CP-scrambling mode is set as shown in Table 7.4-A.

7.1.1 SCRAMBLING RULES

- DES ECB shall be used to scramble copy protected MPEG programs in the POD and to descramble them in the Host. Any residual blocks less than 64 bits in size shall be left in the clear.
- MPEG transport packet headers and adaptation headers shall not be encrypted.
- The MPEG scrambling bits output from the POD shall be set as described in Table 4.3-A.
- CA-scrambled but unauthorized services and CA-scrambled and authorized but unselected services shall pass through POD unaltered, and are therefore useless to the Host.
- CP-scrambling shall only be applied to selected MPEG programs for which EMI is non-zero
- The POD shall CP-scramble only authorized and selected programs. The POD shall immediately switch from rescramble mode to pass-through mode when the active program is deauthorized by the CA System.
- No data shall be double scrambled with both CA and CP-scrambling.

7.2 TRANSPORT PROCESSING

MPEG packet scrambling parity may take on the values 0 or 1 without limitation on CP-scrambled output from the POD. The POD shall set the MPEG packet scrambling bits as defined in section Table 4.3-A.

CP-scrambling mode changes (i.e. transitions from "CP-scrambling ON" to "CP-scrambling OFF/Clear") shall be handled so as to minimize any period of time where copy protected content is sent unscrambled from POD to Host. If the status of CA-scrambling (e.g. due to a change in POD entitlement or a change in the scrambling mode of the MPEG stream input to the POD), then the POD shall alter the mode of its input CA-descrambling prior to altering its mode of output CP- scrambling.

7.3 TIMING OF SCRAMBLING MODE TRANSITIONS

CP-scrambling mode changes from "CP-scrambling OFF" to "CP-scrambling ON" shall be accomplished quickly and in no case more than 1.5 seconds after the event that causes the mode change, e.g., an EMI change from 0 to non-zero. All MPEG packets of the relevant program shall be CP-scrambling as soon as possible following a EMI change from 0 to a protected value of 1, 2, or 3. A change from EMI >0 to EMI=00 shall result in scrambling going inactive within 1.5 seconds.

CA-scrambling may be effected by receipt of encryption management or control messages or by changes in the CA-scrambling mode of the content. The POD shall continue to comply with all CP-scrambling requirements while responding to any such messages or mode changes.

7.4 CP-SCRAMBLING AS A FUNCTION OF CA-SCRAMBLING AND EMI VALUE

The POD shall apply copy protection scrambling of content flowing to the Host as shown in Table 7.4-A.

Table 7.4-A CP-Scrambling based on CA-Scrambled State and EMI Value

CA Scrambling State	EMI Value	POD Scrambles Output	Comments
Unscrambled	00	No	
Unscrambled	01, 10, or 11	No	Undesired*
Scrambled	00	No	
Scrambled	01, 10, or 11	Yes	

* Cable operators should CA-Scramble all programs with non-zero EMI. Only CA-Scrambled programs will be protected from unauthorized copying.

8. HOST, POD, & HEADEND MESSAGING PROTOCOLS

8.1 MESSAGE PROTOCOL OVERVIEW

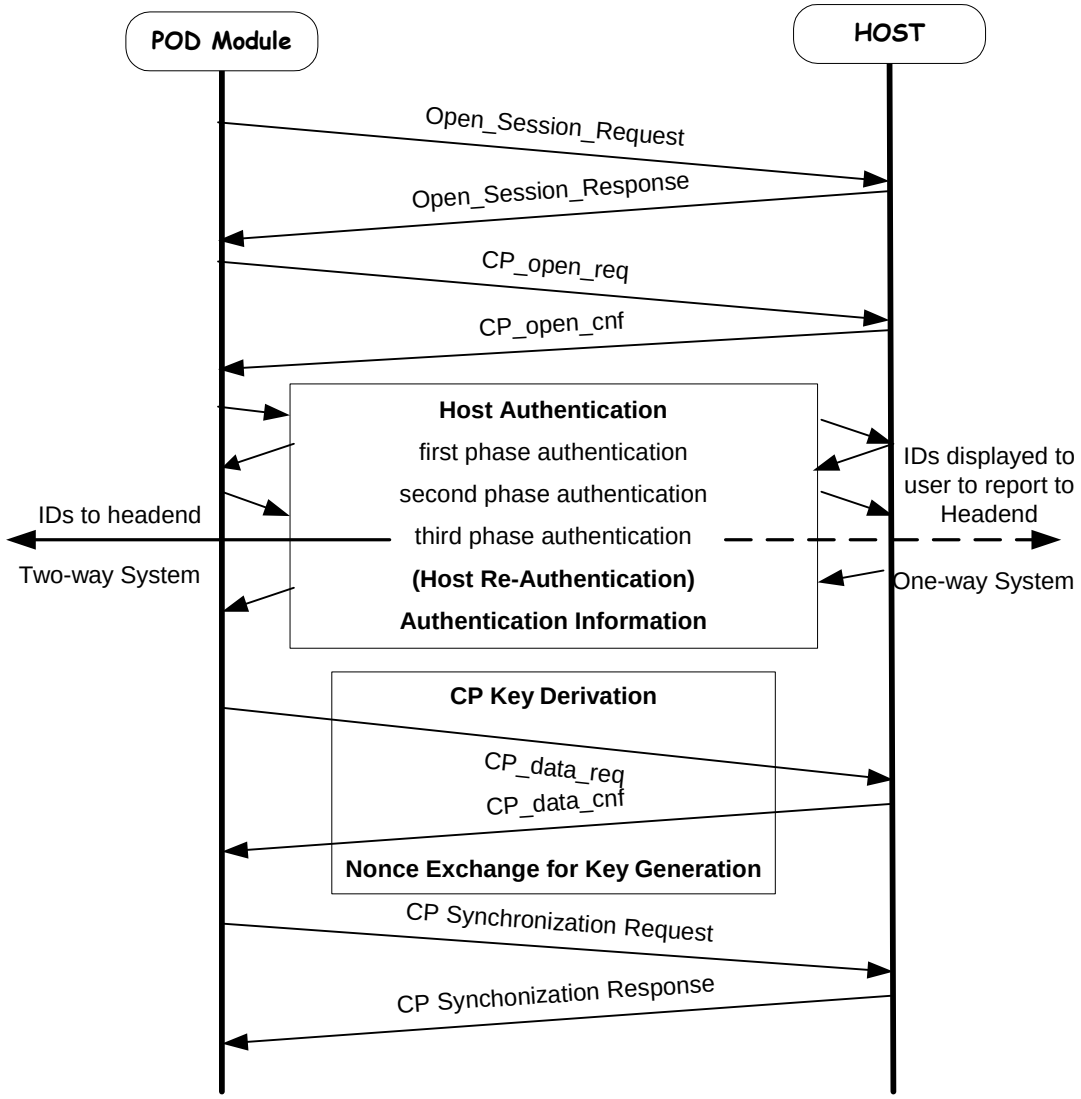

Figure 8.1-A Copy Protection Message Protocol Overview

Figure 8.1-A gives an overview of content copy protection protocol. In this overview, messages start from the POD after the Host recognizes it.

1) The POD module initiates a copy protection (CP) session by sending open_session_request to the Host. An open_session_response is returned by the Host to the POD module in order to allocate a session number. If the request cannot be fulfilled the POD shall treat the Host as if the Host Certificate was invalid.

2) Upon receiving the allocated session number, the POD module checks the copy protection support capability of the Host. The Host responds to the POD with the type of CP System it supports. If the Host does not support the resource defined in Table 8.2-H the POD shall treat the Host as if the Host Certificate was invalid.

3) The POD, if it holds a valid Authentication Key in non-volatile memory, shall attempt to perform a re-authentication procedure by requesting the Host's Authentication Key (second phase authentication.) If the POD's and the Host's Authentication Keys match, and headend ID validation is complete, the protocol can immediately proceed to step 4. If the keys do not match, the POD carries out a full Host authentication, performing all three phases.

4) Once the POD and Host have completed the authentication procedure, new random numbers are exchanged between POD and Host, and a copy protection key can be generated between them. This key is used to scramble content in the POD module and descramble it in the Host.

5) After generating its CP-Key, the POD module notifies the Host about its intention to start to transmit the copy protection data. When the Host is ready (meaning the decryption key has been generated), the Host replies to the POD.

8.2 POD & HOST COMMON MESSAGES

8.2.1 OPENING A SESSION

The POD module requests a session to be opened to a resource on its transport connection. Since Host provides resources it replies directly with a session number in its open session response.

Two objects defined at Session Protocol Data Unit (SPDU) layer, open_session_request() and open_session_response() are used here. Detailed SPDU data structure and other SPDU objects are defined in section 7.2 of CEA-679-C (Part B)

Table 8.2-A Copy Protection Open Session Information

SPDU Tag / Object	Tag Value (Hex)	Action	Direction
Open_Session_Request()	91	The POD module requests a session of the Copy Protection resource to be opened.	POD → Host
Open_Session_Response()	92	The Host responds with a session status. If opened, a session number is assigned. The session number shall then be used for all subsequent exchanges of messages (APDUs) between POD and Host.	POD ← Host

8.2.1.1 OPEN_SESSION_REQUEST( ) SYNTAX

Opening a copy protection request uses an object defined in the Session Layer protocol. The POD module issues this SPDU object to request opening a copy protection session between the POD and Host.

Table 8.2-B Open_Session_Request() Message Syntax

Message Syntax	bits	bytes	Description
open_session_request () {			
Open_session_request_tag	8	1	Has the value of 91 (hex)
Length_field()	8	1	length_field() shall have the following values set: size_indicator = 0, length_value = 4

Resource_identifier()	32	4	Resource_identifier () is defined in CEA-679-C (Part B) section 8.2.2.
			Resource_identifier() {
			resource_id_type 2 bits
			if (resource_id_type != 3) {
			resource_class 14 bits
			resource_type 10 bits
			resource_version 6 bits
			}
			else {
			private_resource_definer 10 bits
			private_resource_identity 20 bits
			}
			}

}			

As specified in ANSI/SCTE 28 2007, the resource_identifier must match in both class and type of resource that the Host has in its list of available resources. Copy protection resource coding is listed in Table 8.2-C.

Table 8.2-C Copy Protection Resource Class

Resource	Class	Type	Version	Identifier
Copy Protection	176	3	1	00B000C1

If the version field of the supplied resource identifier is zero, then the Host shall use the current version in its list. If the version number in the request is less than or equal to the current version number in the Host's list then the current version is used. If the requested version number is higher than the version in the Host's list, the Host shall refuse the request with the appropriate return code as defined in CEA-679-C.

8.2.1.2 OPEN_SESSION_RESPONSE( ) SYNTAX

The Host issues this object to the POD to allocate a session number or to tell the POD that its request could not be met.

Table 8.2-D Open_Session_Response() Message Syntax

Message Syntax	bits	bytes	Description
open_session_response () {			
Open_session_response_tag	8	1	Has the value of 92 (hex)
Length_field()	8	1	length_field() shall have the following values set: size_indicator = 0, length_value = 7
Session_status	8	1	Session status values listed in CEA-679-C, Part B.
Resource_identifier()	32	4	Resource_identifier () is defined Table 8.2-C.
Session_nb	16	2	The Host allocates session number for the requested session. Value 0 is reserved. The session_nb shall be used for all subsequent exchanges of APDUs between the POD module and Host until session is closed. When the requested session could not be opened (session_status != 0), the session_nb has no meaning.
}			

8.2.2 HOST CAPABILITY EVALUATION

The NRSS Copy Protection Framework requires the POD module to check the Host's ability to support the CP System, when the POD module is powered on and before starting the Key Exchange process.

Two objects, CP_open_req() and CP_open_cnf(), as defined at the Application Protocol Data Unit (APDU) layer are used here.

Table 8.2-E Host CP Support Capability Evaluation Messages

APDU Tag / Object	Tag Value (Hex)	Action	Direction
CP_open_req()	9F9000	POD module queries which copy protection system is supported by Host.	POD → Host
CP_open_cnf()	9F9001	Host replies to POD module.	POD ← Host

8.2.2.1 CP_OPEN_REQ( ) SYNTAX

This APDU object is issued by the POD module to query the Host's ability to support various copy protection systems.

Table 8.2-F CP_open_req() Message Syntax

Message Syntax	bits	bytes	Description
CP_open_req () {			
CP_open_req_tag	24	3	Has the value of 9F9000 (hex)
Length_field()	8	1	length_field () is defined in CEA-679-C, Part B, section 7. Since there is no other field followed, length_field() shall have the following values set: size_indicator = 0, length_value = 0
}			

8.2.2.2 CP_OPEN_CNF( ) SYNTAX

This object is issued by the Host to the POD module.

Table 8.2-H defines the value of CP_system_id_bitmask. If system 2 is not supported the POD shall treat the Host as if its certificate was invalid.

Table 8.2-G CP_open_cnf() Message Syntax

Message Syntax	bits	bytes	Description
CP_open_cnf () {			
CP_open_cnf_tag	24	3	Has the value of 9F9001 (hex)
Length_field()	8	1	length_field () is defined in CEA-679-C, Part B, section 7. The length_field() shall have the following values set: size_indicator = 0, length_value = 4
CP_system_id_bitmask	32	4	Values are list in Table 8.2-H.
}			

Table 8.2-H CP_system_id_bitmask Values

CP_system_id_bitmask	Bit Number	Description
System 1	0	reserved
System 2	1	POD CP System
System 3	2	reserved
System 4	3	reserved
System 5	4	reserved

For an example, if bit number 0, 1 and 3 are set to 1, it means that Host has the capability of supporting System 1, System 2, and System 4.

8.2.3 COPY PROTECTION KEY GENERATION

The POD module sends a CP-Key generate request to the Host with the POD_ID and N_module. Upon receipt of this request, the Host shall generate N-Host and send it to the POD along with the Host_ID. Both the POD and the Host shall then generate a new CP-Key, using both nonces and the shared DH private key.

Two objects, CP_data_req() and CP_data_cnf(), as defined at Application Protocol Data Unit (APDU) layer are used here.

Table 8.2-I CP_data in the Transmission Key Generation Messages

APDU Tag / Object	Tag Value (Hex)	Action	Direction
CP_data_req()	9F9002	POD module requests the generation of a new transmission key. This message contains POD_ID and a random nonce *N_module* (CP_system_id = 2, send datatype_id = 6, 12, and receive datatype_id = 5, 11).	POD → Host
CP_data_cnf()	9F9003	Host replies to POD module. The response contains Host_ID and a random nonce (*N_Host*) generated by the Host.	POD ← Host

8.2.3.1 CP_DATA_REQ( ) SYNTAX IN HOST KEY GENERATION

This APDU object is issued by the POD module to send its ID and random nonce to the Host to generate a new CP content key.

Table 8.2-J CP_data_req() Message Syntax In the Key Generation Messages

Message Syntax	bits	bytes	Description
CP_data_req () {			
CP_data_req_tag	24	3	Has the value of 9F9002 (hex)
Length_field()	8	1	length_field () is defined in CEA-679-c, Part B, section 7. size_indicator = 0, length_value = 27
CP_system_id	8	1	Values are listed in Table 8.2-L: CP_system_id = 2
Send_datatype_nbr	8	1	Send_datatype_nbr shall have the value of 2.
For(i=0; i<Send_datatype_nbr; i++) {	(48)	(2*3)	
Datatype_ID	8	1	When i = 0, Datatype_id = 6 (POD_ID)
	8	1	When i = 1, Datatype_id = 12 (N_module)
Datatype_length	16	2	When i = 0, Datatype_length = 0x0008
	16	2	When i = 1, Datatype_length = 0x0008
For (j=0; j<Datatype_length; j++)	(128)	(16)	
{			
Data_type	64	8	When i = 0, Data_type = *POD_ID*
	64	8	When i = 1, Data_type = N_module;
}			
}			
Request_datatype_nbr	8	1	Request_datatype_nbr shall have the value of 2.
For(i=0; i<Request_datatype_nbr; i++)	(16)	(2*1)	
{			
Datatype_id	8	1	When i = 0, Datatype_id = 5 (Host_ID)
	8	1	When i = 1, Datatype_id = 11 (N_Host)
}			
}			

Table 8.2-K defines data type ID values and data type parameter size.

Table 8.2-K Datatype_ID and Datatype_length Values

Datatype_id	id value	Size (Bytes)
Manufacturer_id	1	50 (Maximum)
Reserved	2	
Reserved	3	
Reserved	4	
Host_ID	5	5
POD_ID	6	8
Host_ManCert (Host Manufacturer XCA Certificate)	7	2048*
POD_ManCert (POD Manufacturer XCA Certificate)	8	2048*
Reserved	9	
Reserved	10	
N_Host (Host's challenge to POD)	11	8
N_module (POD's challenge to Host)	12	8
DH_pubKey_H (Host DH Public Key)	13	128
DH_pubKey_P (POD DH Public Key)	14	128
Host_DevCert (Host Device Certificate Data)	15	2048*
POD_DevCert (POD Device Certificate Data)	16	2048*
$SIGN_H$(the signature of Host DH public key)	17	128
$SIGN_P$(the signature of POD DH public key)	18	128
CCI_N_host	19	8
Reserved	20	
Reserved	21	
$AuthKey_H$ (Host Authentication Key)	22	20
$AuthKey_P$ (POD Authentication Key)	23	20
CCI_N_module	24	8
CCI_data	25	1
Program_Number	26	2
CCI_auth	27	20
CCI_ack	28	20

* Certificates shorter than 2048 shall be padded to 2048 bytes by adding NULL bytes (0x00) at the trailing end. For example a 2000 byte certificate would be padded to 2048 bytes by adding 48 trailling NULL bytes.

Table 8.2-L CP_system_id Values

CP_system_id	ID Value (Binary)
No compatible CP system supported	XXX0 0000
System 1	XXX0 0001
System 2	XXX0 0010
Systems 3 to 30	XXX0 0011 to XXX1 1110
System 31	XXX1 1111
Message is Encrypted	1XXX XXXX
Message is Not Encrypted	0XXX XXXX

Table 8.2-L, above, defines CP_system_id values. The POD CP System is "System 2".

8.2.3.2 CP_DATA_CNF( ) SYNTAX IN KEY GENERATION

This object also contains the Host_ID and nonce so the POD module can derive its CP-Key.

Table 8.2-M CP_data_cnf() Message Syntax In the Key Generation Messages

Message Syntax	bits	bytes	Description
CP_data_cnf () {			
CP_data_cnf_tag	24	3	Has the value of 9F9003 (hex)
Length_field()	8	1	length_field () is defined in CEA-679-C, part B, section 7. The length_field() shall have the following values set: size_indicator = 0, length_value = 21
CP_system_id	8	1	Values are listed in Table 8.2-L.
Send_datatype_nbr	8	1	Send_datatype_nbr shall have the value of 2.
For(i=0; I<Send_datatype_nbr; i++)	(48)	(2*3)	
{			
Datatype_id	8	1	When i = 0, Datatype_id= 5 (Host_ID)
	8	1	When i = 1, Datatype_id=11 (N_Host)
Datatype_length	16	2	When i = 0, Datatype_length = 0x0005
	16	2	When i = 1, Datatype_length = 0x0008
For (j=0; j<Datatype_length; j++)	(104)	(13)	
{			
Data_type	40	5	When i = 0, Data_type = Host_ID
	64	8	When i = 1, Data_type = N_Host
}			
}			
}			

8.2.4 HOST AND POD SYNCHRONIZATION

The NRSS Copy Protection Framework requires POD module to notify the Host about its intention to start to transmit the copy protection data. When Host is ready, Host needs to reply to the POD module. Two objects, CP_sync_req() and CP_sync_cnf(), as defined at Application Protocol Data Unit (APDU) layer are used here.

Table 8.2-N Host and POD module Synchronization Messages

APDU Tag / Object	Tag Value (Hex)	Action	Direction
CP_sync_req()	9F9004	The POD module notifies the Host when it is ready to start to transmit the CP data.	POD → Host
CP_sync_cnf()	9F9005	Host replies to the POD module to confirm Host readiness.	POD ← Host

8.2.4.1 CP_SYNC_REQ() SYNTAX

This object is issued by the POD module to tell the Host that it is ready to send copy protected data and to check if Host is ready.

Table 8.2-O CP_sync_req() Message Syntax

Message Syntax	Bits	bytes	Description
CP_sync_ req () {			
CP_sync_req_tag	24	3	Has the value of 9F9004 (hex)
Length_field()	8	1	length_field () is defined in CEA-679-C, part B, section 7. The length_field() shall have the following values set: size_indicator = 0, length_value = 0
}			

8.2.4.2 CP_SYNC_CNF() SYNTAX

The CP_sync_cnf() object is issued by the Host to tell POD that Host is ready to accept copy protected data.

Table 8.2-P CP_sync_cnf() Message Syntax

Message Syntax	bits	bytes	Description
CP_sync_ cnf () {			
CP_sync_req_tag	24	3	Has the value of 9F9005 (hex)
Length_field()	8	1	length_field () is defined in CEA-679-C, part B, section 7. The length_field() shall have the following values set: size_indicator = 0, length_value = 1
Status_field	8	1	Values are listed in Table 8.2-Q
}			

Status_field shall return the status of the ***CP_sync_req().*** If the Host is ready to receive the incoming stream, then ***Status_field*** shall be set to 0x00. Otherwise, it shall be set to one of the values indicated in Table 8.2-Q.

Table 8.2-Q Status_field Value

Status_field	Value
OK	00
Error – No CP support	01
Error – Host Busy	02
Reserved	03-FF

8.3 POD-HOST CP MESSAGE PROTOCOL

8.3.1 PROTOCOL FLOW OVERVIEW

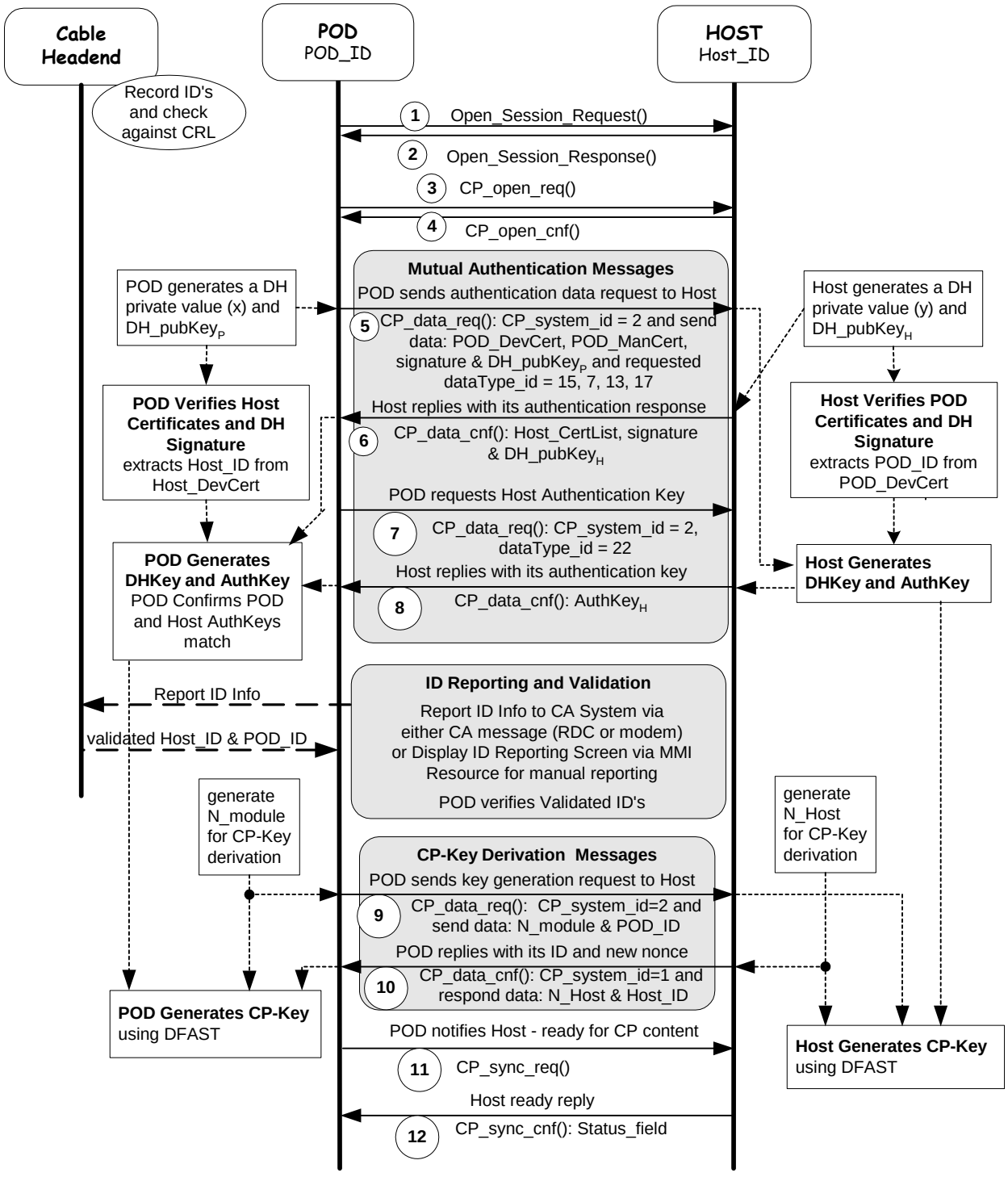

Figure 8.3-A POD-Host CP Message Protocol Overview

Figure 8.3-A gives an overview of the CP System message flow protocol. Full authentication is shown, in which the POD does not have a valid authentication key at power-up. If the POD does hold a valid authentication key, message flow would advance from the CP_open_cnf to "POD requests Host Authentication Key." Message flow would then proceed to the "CP-Key Derivation Messages" if the POD and Host Authentication Keys matched. If the keys did not match, message flow would go back to the beginning of "Host Authentication Messages."

Table 8.3-A POD-Host CP Message Reference Sections

#	Message Name	Protocol Layer / Tag Value (hex)	Reference Section	Purpose
1	Open_Session_Request	SPDU / 91	Section 8.2.1.1	Open CP session
2	Open_Session_Response	SPDU / 92	Section 8.2.1.2	
3	CP_open_req	APDU / 9F9000	Section 8.2.2.1	Evaluate Host
4	CP_open_cnf	APDU / 9F9001	Section 8.2.2.2	
5	CP_data_req	APDU / 9F9002	Section 8.3.2.1	POD & Host authentication data
6	CP_data_cnf	APDU / 9F9003	Section 8.3.2.2	
7	CP_data_req	APDU / 9F9002	Section 8.3.3.1	Authentication Key verification
8	CP_data_cnf	APDU / 9F9003	Section 8.3.3.2	
9	CP_data_req	APDU / 9F9002	Section 8.2.3.1	CP-Key derivation
10	CP_data_cnf	APDU / 9F9003	Section 8.2.3.2	
11	CP_sync_req	APDU / 9F9004	Section 8.2.4.1	POD & Host Synchronization
12	CP_sync_cnf	APDU / 9F9005	Section 8.2.4.2	

8.3.1.1 AUTHENTICATION PROTOCOL IMPLEMENTATION

The POD Module initiates the Host authentication protocol after a CP session is open and the Host's ability to support copy protection has been evaluated by POD. The Host authentication is achieved in a three-step process. The first step of authentication is based on POD being able to verify the Host certificate list signatures and the Host being able to verify the POD certificate list signatures. The second step of authentication is based on the POD being able to verify that its Authentication Key is the same as the one computed by the Host. The third step of authentication is based on the headend verifying that the Host_ID and POD_ID are not included in CRLs, and the POD confirm the Validated ID's received from the headend are the same as those stored locally.

At power-up, if the POD holds a valid Authentication Key, it shall utilize the second step of authentication described above to attempt to perform a re-authentication (steps 17 & 18 below.) If the POD is able to verify that its Authentication Key is the same as the one stored in the Host, authentication is complete. When headend ID validation is also complete the POD can proceed with CP-Key generation. If the keys do not match, then a full Host authentication shall be performed.

8.3.1.2 POD CPS PROTOCOL STEPS - FULL AUTHENTICATION

1. The POD Module initiates the authentication protocol by sending a challenge request to the Host. The POD generates a secret random x, $1 \leq x \leq n-2$, and sends the Host a message. This message contains the POD certificate list (*POD_DevCert* and *POD_ManCert*), a signature of the Diffie-Hellman public key, and the Diffie-Hellman public key DH_pubKey_P. The request is implemented by the *CP_data_req()* object, as defined in APDU layer. The *CP_data_req()* message used here is detailed in section 8.3.2.1.
2. After receiving *CP_data_req()*, Host generates a secret random y, $1 \leq y \leq n-2$, and sends its reply with its Host Certificate List (*Host_DevCert and Host_ManCert*), a signature of the Host Diffie-Hellman public key, and the Diffie-Hellman public *key* DH_pubKey_H to the POD Module. This response is implemented by the object *CP_data_cnf()* object, as detailed in section 8.3.2.2.
3. The POD verifies the Host Certificate List by:
   - Checking the value of the certificate type or format field; and
   - Verifing the X.509 certificate chain that includes *Host_DevCert*, *Host_ManCert*, and the PHICA root certificate, which is resident in the POD.
4. The Host verifies the POD certificate by:
   - Checking the value of the certificate type or format field; and
   - Verifing the the X.509 certificate chain that includes *POD_DevCert, POD_ManCert*, and the PHICA root certificate which is resident in the Host.
5. If *Host_DevCert* is valid, then the POD extracts Host_ID from the Host device certificate.
6. If *POD_DevCert* is valid, then the Host extracts the POD_ID from it.
7. The POD extracts the Host's public key from the Host Certificate, and then uses it to verify the signature: $SIGN_H$ (DH_pubKey_H)
8. The Host extracts the POD's public key from the POD_device certificate, and then uses it to verify the signature: $SIGN_P$ (DH_pubKey_P).
9. The POD and the Host verify the RSA signature on these received messages and this proves that the messages were signed using the appropriate private key.
10. The Host computes the DH shared secret key *DHKey* from its private exponent, *y*, and DH_pubKey_P and then calculates its Authentication Key $AuthKey_H$ based on *DHKey, POD_ID* and *Host_ID* as described in section 4.1 ,.
11. The POD computes the DH shared secret key *DHKey* from its private exponent, *x*, and DH_pubKey_H and then calculates its Authentication Key $AuthKey_P$ based on *DHKey, POD_ID* and *Host_ID* as described in section 4.1.
12. The POD sends a message to Host to request the Authentication Key $AuthKey_H$ computed by the Host using*CP_data_req()* as detailed in section 8.3.3.1.
13. The Host sends its response $AuthKey_H$ to the POD Module by using the message *CP_data_cnf()*. This response message is detailed in section 8.3.3.2.
14. The POD compares *AuthKeyP* to *AuthKeyH*. If they match the POD continues with CP-Key derivation; otherwise the POD shall respond to failure of the CP system as described in section 3.2.2.

15. The POD sends the POD and Host ID information to the headend by one of two means:
    A. If no automated reporting path is established:
        a. The POD Module opens an MMI dialog and displays the POD_ID and Host_ID to the subscriber with a telephone number for manual reporting.
        b. The end-user will contact the service provider and report the ID information; see section 3.2.5.1.
    B. If an automated means is established, e.g., the RDC or a telco modem, the POD shall use it to send the ID information to the headend.
16. (Cable Headend CRL Checking: second step of authentication) Check if Host_ID and POD_ID are in the headend CRLs. Validate the Host/POD ID's. This check may not occur in real time; see section 3.2.5.1.
17. (Cable Headend - > POD) Send EMM to authorize the POD; see section 3.2.5.1.
18. (Cable Headend - > POD) Headend sends validated IDs back to the POD Module in an authenticated CA system message. See section 3.2.5.1.
19. The POD shall authenticate the message and verify the validated ID's match the stored device ID's.

8.3.2 HOST AUTHENTICATION MESSAGES

Two objects, CP_data_req() and CP_data_cnf(), as defined at Application Protocol Data Unit (APDU) layer are used to exchange the authentication messages.

Table 8.3-B Host Authentication Messages

APDU Tag / Object	Tag Value (Hex)	Action	Direction
CP_data_req()	9F9002	POD module sends its authentication data to Host.	POD →Host
CP_data_cnf()	9F9003	Host replies to POD module.	POD ← Host

8.3.2.1 CP_DATA_REQ( ) SYNTAX IN HOST AUTHENTICATION REQUEST MESSAGE

This APDU object is issued by the POD module to send its authentication data to the Host. The POD Certificate Data (*POD_DevCert* and *POD_ManCert*),a signature of the POD Diffie-Hellman public key, and the Diffie-Hellman public key (*DH_pubKey*$_P$) are included in this message.

POD - > Host:

DH_pubKey$_P$, SIGN$_P$ (DH_pubKey$_P$), POD_DevCert, POD_ManCert.

Table 8.3-C CP_data_req in the Host Authentication Request Message

Message Syntax	bits	bytes	Description
CP_data_req () {			
CP_data_req_tag	24	3	Has the value of 9F9002 (hex)
Length_field()	24	3	Defined by and with values set to:
			Size_indicator = 1 (1 bit, bslbf);
			Length_field_size = 2 (7 bits, uimsbf);
			Length_value_byte[0] = 17 (8 bits, bslbf); (most significant byte)
			Length_value_byte[1] = 19 (8 bits, bslbf); (least significant byte)
			(message size = 4371 bytes from CP_system_ID)
CP_system_id	8	1	CP_system_id = 2 (POD CPS)
Send_datatype_nbr	8	1	Send_datatype_nbr shall have the value of 4
For(i=0; i<Send_datatype_nbr; i++) {	(96)	(12)	
Datatype_ID	8	1	When i = 0, Datatype_ID has the value of 16 (POD_DevCert);
	8	1	
	8	1	When i = 1, Datatype_ID has the value of 8 (POD_ManCert);
	8	1	When i = 21, Datatype_ID has the value of 14 (DH_pubKey_P);
			When i = 3, Datatype_ID has the value of 18 ($SIGN_P$);
Datatype_length	16	2	When i = 0, Datatype_length has the value of 2048;
	16	2	When i = 1, Datatype_length has the value of 2048;
	16	2	When i = 2, Datatype_length has the value of 128;
	16	2	When i = 3, Datatype_length has the value of 128;
For (j=0; j<Datatype_length; j++) {		(4352)	
Data_type	16384	2048	When i = 0, Data_type = POD_DevCert;
	16384	2048	When i = 1, Data_type = POD_ManCert;
	1024	128	When i = 2, Data_type = DH_pubKey_P;
	1024	128	When i = 3, Data_type = $SIGN_P$;
}			
}			
Request_datatype_nbr	8	1	Request_datatype_nbr shall have the value of 4.
For(i=0; i<Request_datatype_nbr; i++) {	(32)	(4)	
Datatype_ID	8	1	When i = 0, Datatype_ID has the value of 15 (Host_DevCert)
	8	1	When i = 1, Datatype_ID has the value of 7 (Host_ManCert)
	8	1	When i = 2, Datatype_ID has the value of 13 (DH_pubKey_H).
	8	1	When i = 3, Datatype_ID has the value of 17 ($SIGN_H$).
}			
}			

8.3.2.2 CP_DATA_CNF() SYNTAX IN HOST AUTHENTICATION RESPONSE MESSAGE

This APDU object is issued by the Host to send its response data to the POD. Host's certificate list(*Host_DevCert and Host_ManCert*), a signature of the POD and Host Diffie-Hellman public key, and Diffie-Hellman public key (*DH_pubKey$_H$*) are included in this message.

Host -> POD:

DH_pubKey$_H$, SIGN$_H$ (DH_pubKey$_H$), Host_DevCert, Host_ManCert

Table 8.3-D CP_data_cnf in the Host Authentication Response Message

Message Syntax	bits	bytes	Description
CP_data_cnf () {			
CP_data_cnf_tag	24	3	Has the value of 9F9003 (hex)
Length_field()	24	3	Defined by and with values set to:
			Size_indicator = 1 (1 bit, bslbf);
			Length_field_size = 2 (7 bits, uimsbf)
			Length_value_byte[0] = 17 (8 bits, bslbf) (most significant byte)
			Length_value_byte[1] = 14 (8 bits, bslbf) (least significant byte)
			(message size = 4366 bytes after length bytes)
CP_system_id	8	1	CP_system_id = 2 (POD CPS)
Send_datatype_nbr	8	1	Send_datatype_nbr shall have the value of 4
For(i=0; i<Send_datatype_nbr; i++) {	(96)	(12)	
Datatype_ID	8	1	When i = 0, Datatype_ID has the value 15 (Host_DevCert);
	8	1	When i = 1, Datatype_ID has the value 7 (Host_ManCert);
	8	1	When i = 2, Datatype_ID has the value 13 (DH_pubKey$_H$);
	8	1	When i = 3, Datatype_ID has the value 17 (SIGN$_H$);
Datatype_length	16	2	When i = 0, Datatype_length has the value of 2048
	16	2	When i = 1, Datatype_length has the value of 2048
	16	2	When i = 2, Datatype_length has the value of 128
	16	2	When i = 3, Datatype_length has the value of 128
For (j=0; j<Datatype_length; j++) {	34816	(4352)	
Data_type	16384	2048	When i = 0, Data_type = Host_DevCert;
	16384	2048	When i = 1, Data_type = Host_ManCert;
	1024	128	When I = 2, Data_type = DH_pubKey$_H$;
	1024	128	When i = 3, Data_type = SIGN$_H$
}			
}			
}			

8.3.3 HOST AUTHENTICATION KEY VERIFICATION MESSAGES

Two objects, CP_data_req() and CP_data_cnf(), as defined at Application Protocol Data Unit (APDU) layer, are used for the POD module to obtain the authentication key from the Host.

Table 8.3-E Host Authentication Key Verification Messages

APDU Tag / Object	Tag Value (Hex)	Action	Direction
CP_data_req()	9F9002	POD module requests Host authentication key.	POD → Host
CP_data_cnf()	9F9003	Host replies to POD module.	POD ← Host

8.3.3.1 CP_DATA_REQ() IN THE AUTHENTICATION KEY VERIFICATION REQUEST MESSAGE

This APDU object is issued by the POD module to send its authentication key request to the Host.

Table 8.3-F CP_data_req in the Authentication Key Verification Request Message

Message Syntax	bits	bytes	Description
CP_data_req () {			
CP_data_req_tag	24	3	Has the value of 9F9002 (hex)
length_field()	8	1	Length_field () is defined CEA-679-C, part B, section 7. The length_field() in this message shall have the following values set: size_indicator = 0, length_value = 4
CP_system_id	8	1	CP_system_id = 2
Send_datatype_nbr	8	1	Send_datatype_nbr shall have the value of 0.
Request_datatype_nbr	8	1	Request_datatype_nbr shall have the value of 1.
For(i=0; i<Request_datatype_nbr; i++) {	(8)	(1)	
Datatype_ID	8	1	Datatype_ID has the value of 22 (*AuthKey*$_H$, see Table 8.2-K).
}			
}			

8.3.3.2 CP_DATA_CNF() IN THE AUTHENTICATION KEY VERIFICATION RESPONSE MESSAGE

This APDU object is issued by the **Host** to send its authentication key ($AuthKey_H$) to the POD.

Table 8.3-G CP_data_cnf in the Authentication Key Verification Response Message

Message Syntax	bits	bytes	Description
CP_data_cnf () {			
CP_data_cnf_tag	24	3	Has the value of 9F9003 (hex)
length_field()	8	1	length_field () is defined in CEA-679-C, part B, section 7. The length_field() in this message shall have the following values set: size_indicator = 0, length_value = 25
CP_system_id	8	1	CP_system_id = 2
Send_datatype_nbr	8	1	Send_datatype_nbr shall have the value of 1.
For(i=0; i<Send_datatype_nbr; i++) {	(16)	(2)	
Datatype_ID	8	1	Datatype_ID has the value of 22 ($AuthKey_H$, see Table 8.2-K)
Datatype_length	16	2	Datatype_length has the value of 20 (see Table 8.2-K)
For (j=0; j<Datatype_length; j++) {			
Data_type	160	20	Data_type = $AuthKey_H$ (see Table 8.2-K)
}			
}			
}			

8.4 CCI SIMPLE AUTHENTICATION TUNNEL PROTOCOL (SATP) MESSAGES

The simple authentication tunnel protocol is a two-pass protocol. First, keys required by the SATP are generated and passed. Second, CCI is transmitted to the Host with a fingerprint appended to the CCI byte. In detail, the POD generates a nonce and sends it in a request message to the Host to generate a nonce. The Host generates a nonce and sends it back in a reply message. Then the POD calculates a fingerprint using the CCI value, program number and each nonce and sends the CCI value with the fingerprint appended in a data request message. Finally, the Host sends a reply message without a data payload.

Table 8.4-A CCI Simple Authentication Tunnel Protocol Messages

APDU Tag / Object	Tag Value (Hex)	Action	Direction
CP_data_req	9F9002	POD module requests the generation of a new 8 byte random number. The message contains the random nonce generated by the POD (*CCI_N_module*) and the program number (*program_number*), the same one found in the CA_pmt_req() message. (CP_system_id = 2, send datatype_id = 24, 26, and request datatype_id = 19, 26).	POD → Host
CP_data_cnf	9F9003	Host replies to POD module with the requested data types. The response contains the random nonce gernerated by the Host (*CCI_N_Host*) and the program number (*program_number*). (CP_system_id = 2, send datatype_id = 19, 26).	POD ← Host
CP_data_req	9F9002	POD module sends the CCI payload (*CCI_data*), the program number (*program_number*) and the calculated message digest (*CCI_auth).*	POD → Host

		(CP_system_id = 2, send datatype_id = 25, 26, 27, request datatype_id=26, 28).	
CP_data_cnf	9F9003	Host replies to POD module with CCI_ack. (CP_system_id = 2, send datatype_id=26, 28)	POD ← Host

Table 8.4-B CP_data_req() Message Syntax in SATP Key Generation

Message Syntax	bits	bytes	Description
CP_data_req(){			
CP_data_req_tag	24	3	Has the value of 0x9F9002.
length_field()	8	1	Has the value of 0x15. size_indicator = 0, length_value =21
CP_system_id	8	1	Has the value of 2. Values are listed in Table 8.2-L.
Send_datatype_nbr	8	1	Has the value of 2.
for(i=0; i<Send_datatype_nbr; i++)			
{			
Datatype_id	8	1	i = 0, Datatype_id has the value of 24 (*CCI_N_module*).
	8	1	i = 1, Datatype_id has the value of 26 (*program_number).*
Datatype_length	16	2	i = 0, Datatype_length has the value of 0x0008.
	16	2	i = 1, Datatype_length has the value of 0x0002.
for (j=0; j<Datatype_length; j++)			
{			
Data_type	64	8	When i = 0, Data_type = *CCI_N_module.*
	16	2	When i = 1, Data_type = *program_number*.
}			
}			
Request_datatype_nbr	8	1	Has the value of 2.
for(i=0; i<Request_datatype_nbr; i++)			
{			
Datatype_id	8	1	When i=0, Datatype_id has the value 19 (*CCI_N_Host*).
	8	1	When i=1, Datatype_id has the value 26 (*program_number).*
}			
}			

Table 8.4-C CP_data_cnf() Message Syntax in CCI SATP Key Generation

Message Syntax	bits	bytes	Description
CP_data_cnf(){			
CP_data_cnf_tag	24	3	Has the value of 0x9F9003.
length_field()	8	1	Has the value of 0x12. size_indicator = 0, length_value = 18
CP_system_id	8	1	Has the value of 2. Values are listed in Table 8.2-L
Send_datatype_nbr	8	1	Has the value of 2.
for(i=0; i<Send_datatype_nbr; i++)			
{			
Datatype_id	8	1	i = 0, Datatype_id has the value of 19 (*CCI_N_Host*).
	8	1	i = 1, Datatype_id has the value of 26 (*program_number*).
Datatype_length	16	2	i = 0, Datatype_length has the value of 0x0008.
	16	2	i = 1, Datatype_length has the value of 0x0002.
for (j=0; j<Datatype_length; j++)			
{			
Data_type	64	8	When i = 0, Data_type = *CCI_N_Host.*
	16	2	When i = 1, Data_type = *program_number*.
}			
}			
}			

Table 8.4-D CP_data_req() Message Syntax in CCI SATP Transmission

Message Syntax	bits	bytes	Description
CP_data_req(){			
CP_data_req_tag	24	3	Has the value of 0x9F9002.
length_field()	8	1	Has the value of 0x23. size_indicator = 0, length_value = 35
CP_system_id	8	1	Has the value of 2. Values are listed in Table 8.2-L.
Send_datatype_nbr	8	1	Has the value of 3.
for(i=0; i<Send_datatype_nbr; i++)			
{			
Datatype_id	8	1	i = 0, Datatype_id has the value of 25 (*CCI_data*).
	8	1	i = 1, Datatype_id has the value of 26 (*program_number*).
	8	1	i = 2, Datatype_id has the value of 27 (*CCI_auth*).
Datatype_length	16	2	i = 0, Datatype_length has the value of 0x0001
	16	2	i = 1, Datatype_length has the value of 0x0002
	16	2	i = 2, Datatype_length has the value of 0x0014
for (j=0; j<Datatype_length; j++)			
{			
Data_type	8	1	When i = 0, Data_type = *CCI_data.*
	16	2	When i = 1, Data_type = *program_number.*
	160	20	When i = 1, Data_type = *CCI_auth.*
}			
}			
Request_datatype_nbr	8	1	Has the value of 2.
for(i=0; i<Request_datatype_nbr; i++)			
{			
Datatype_id	8	1	When i=0, Datatype_id has the value 28 (*CCI_ack*).
Datatype_id	8	1	When i=0, Datatype_id has the value 26 (program_number).
}			
}			

Table 8.4-E CP_data_cnf() Message Syntax in CCI SATP Transmission

Message Syntax	bits	bytes	Description
CP_data_cnf(){			
CP_data_cnf_tag	24	3	Has the value of 0x9F9003.
length_field()	8	1	Has the value of 0x1E, size_indicator = 0, length_value = 30
CP_system_id	8	1	Has the value of 2. Values are listed in Table 8.2-L.
Send_datatype_nbr	8	1	Has the value of 2.
for(i=0; i<Send_datatype_nbr; i++)			
{			
Datatype_id	8	1	i = 0, Datatype_id has the value of 28 (*CCI_ack*).
	8	1	i = 1, Datatype_id has the value of 26 (*program_number*).
Datatype_length	16	2	i = 0, Datatype_length has the value of 0x0014.
	16	2	i = 1, Datatype_length has the value of 0x0002.
for (j=0; j<Datatype_length; j++)			
{			
Data_type	160	20	When i = 0, Data_type = *CCI_ack.*
	16	2	When i = 1, Data_type = *program_number.*
}			
}			
}			

Appendix A. Luhn Check Digit (Normative)

The Luhn check digit is calculated over decimal values using the following algorithm.

1. Convert the value into the appropriate decimal format (see Section 3.2.1).
2. Double the value of alternate digits beginning with the first right hand digit (least significant digit) and moving left.
3. Add the individual digits comprising the products obtained in step 2 to each of the unaffected digits in the original number.
4. Subtract the total obtained in step 3 from the next higher number ending in 0. This is equivalent to calculating the "tens complement" of the low order digit of the total. If the total obtained in step 3 is a number ending in 0, then the check digit is 0.

Example:

For the 40-bit Host_ID 0x01 3B2C 021F (hexadecimal), made up from decimal manufacturer number 004 and Unit ID 992,739,871:

1. Concatenate to the 12-digit decimal value 004,992,739,871 per Section 3.2.1.

2. Separate this decimal number into odd and even digits starting from the right (least significant digit):
   digit #: $^{12}{}_{11}{}^{10}$987654321
   'odd' digits: 0,9,2,3,8,1
   'even' digits: 0,4,9,7,9,7

3. Multiply each 'odd' digit by 2:
   0, 9, 2, 3, 8, 1 $\rightarrow$ 0, 18, 4, 6, 16, 2

4. Add the 'even' digits and each individual digit of the products above:
   [0+4 + 9 + 7 + 9 + 7] + [0+1 + 8 + 4 + 6 + 1 + 6 + 2] = 64

5. Subtract the least significant digit of this sum from 10 to form the check digit:
   10 - 4 = 6

The Luhn check digit for this example is "6".

Appendix B. Applying CP-Key to DES Engine (Normative)

METHOD OF APPLICATION

The cryptographic key is applied to many DES engines as a 64-bit value as described in FIPS-PUB 46-2 and FIPS-PUB 81. The POD-CP specification above defines generation of a 56-bit integer CP-Key (Ks_dfast). The 64-bit key is generated from Ks_dfast by adding a parity bit to each 7-bit block.

Starting with Ks_dfast in a 56-bit format:

Ks_dfast = $K_1\ K_2\ K_3 \ldots K_{56}$ Where K_1 represents the most significant bit of Ks_dfast.

By adding parity bits after each 7 bits of Ks_dfast we get the 64-bit key:

K_{64bit} = $K_1\ K_2 \ldots K_7\ \mathbf{P_1}\ \ K_8 \ldots K_{14}\ \mathbf{P_2}\ \ \ldots\ \ \ldots\ \ K_{50} \ldots K_{56}\ \mathbf{P_8}$

where P_i shall be either 0 or 1 so that each octet has odd parity (i.e. there is an odd number of "1" bits).

For example, for an original value of CP-Key:

Ks_dfast= $0123456789abcd_{(16)}$
= 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 $1101_{(2)}$

Break it into eight 7-bit blocks:

Ks_dfast= 0000000 1001000 1101000 1010110 0111100 0100110 1010111 $1001101_{(2)}$

Add the parity bits as the last bit of each octet to get the 64-bit key:

K_{64bit} = 00000000 10010001 11010000 10101101 01111001 01001100 10101110 $10011011_{(2)}$
= $0191d0ad794cae9b_{(16)}$

EXAMPLES OF CP ENCRYPTION OF MPEG DATA IN TRANSPORT PACKETS

This section shows examples of packets before and after DES encryption by the copy protection system. The encryption key used here is **0123456789ABCDEF**$_{(16)}$ in 64-bit format (or 00451338957377$_{(16)}$ in 56-bit format), which is shown in FIPS-PUB 81 as an example. The lines "C:" and "E:" for each example show the transport packet data before and after CP encryption respectively (cleartext and encrypted).

Example 1: A null packet.

C: 47 1f ff 10 ff ff ff ff ff ff ff ff ff ff ff ff ...
E: 47 1f ff 10 ff ff ff ff ff ff ff ff ff ff ff ff ...

CP encryption leaves the packets that don't belong to a copy protected program unchanged.

Example 2: A packet without adaptation field that belongs to a copy protected program.

C: 47 10 22 **1**c **d4 75 09 40 c3 61 ec 26 1a 30 cf 1c c6 e1 d0 d1** ...
E: 47 10 22 **d**c **03 f9 77 f6 89 01 4a 9f 09 f0 ef bc 85 58 9f 9f** ...

DES encryption starts right after the packet header. **transport_scrambling_control** field is changed from **00** to **11** (4th byte: **1**c to **d**c). This field could be either 10 (even key) or 11(odd key) when seamless key refresh mechanism is introduced as per ECN-00075. Each 8-byte block in the packet payload is encrypted with DES ECB mode.

Example 3: A packet with adaptation field that belongs to a copy protected program.

C: 47 00 50 32 02 00 ff 88 f5 32 3e ac 87 eb 10 ...
... c3 d6 88 f7 32 32 ac af eb e0 78 41 11 (end of packet)
E: 47 00 50 f2 02 00 ff bb 5a ec 14 56 8b 66 b4 ...
... 80 50 cf cd ad 7e d1 de eb e0 78 41 11 (end of packet)

DES encryption starts after the adaptation field, which takes 3 bytes in this example (1 byte for adaptation_field_length and 2 bytes for the body). The payload is encrypted the same way except for the short block (5 bytes) at the end, which remains clear. transport_scrambling_control field is changed as described in example 2 above.