

Superseded by New Specification CCCP2.0

OpenCable™ POD Copy Protection System

IS-POD-CP-INT05-010515

INTERIM
SPECIFICATION

5/15/01

Notice

This specification is furnished by Cable Television Laboratories, Inc., as part of the OpenCable™ process on an "AS IS" basis. CableLabs does not provide any representation or warranty, express or implied, regarding its accuracy, completeness, or fitness for a particular purpose. **This specification is copyrighted by CableLabs and, to the best of CableLabs' knowledge, does not contain any intellectual property of third parties. Anyone designing, manufacturing, distributing, or servicing products, or providing services, based upon this specification, may be required to obtain intellectual property licenses from third parties (e.g., MPEG) for technology referenced in this specification.** CableLabs shall have no liability for any party's implementation of this specification without any such required licenses.

© Copyright 2001 Cable Television Laboratories, Inc.

All rights reserved.

Document Status Sheet

Document Control				
Number:	IS-POD-CP-INT05-010515			
Document Title:	OpenCable POD Copy Protection System			
Revision History:	Revision derived from IIS-POD-INT04-010307			
Date:	May 15, 2001			
Responsible Editor:	Jeff Hamilton			
Status:	Work in Progress	Draft	Interim	Released
Distribution Restrictions:	author only	CL/Member	CL/Member/Vendor	Public

Key to Document Status Codes:

Work in Progress	An incomplete document, designed to guide discussion and generate feedback, that may include several alternative requirements for consideration.
Draft	A document in specification format considered largely complete, but lacking review by Members and vendors. Drafts are susceptible to substantial change during the review process.
Interim	A document which has undergone rigorous Member and vendor review, suitable for use by vendors to design in conformance to and for field testing.
Released	A stable document, reviewed, tested and validated, suitable to enable cross-vendor interoperability.

Contents

1.	INTRODUCTION.....	1
1.1	Purpose.....	2
1.2	Revision History.....	3
1.3	References	3
1.4	Acronyms and Abbreviations.....	4
1.5	Copy Protection System Components	5
1.6	Implementation Outline.....	5
1.7	Document Links [Informative].....	6
2.	SYSTEM OVERVIEW.....	7
2.1	NRSS Copy Protection Framework.....	7
2.2	Host Authentication	7
2.2.1	Bi-Directional Host and Cable System	8
2.2.2	Manual Return Authentication	8
2.2.3	Manual Return Authentication – Error and Other Conditions.....	9
2.3	Key Exchange and Interface Encryption.....	10
2.3.1	Setup Phase	11
2.3.2	Copy Control Information.....	11
2.3.3	Key Derivation Phase	11
2.3.4	Multiple Host Support	12
2.3.5	Interface Encryption	12
2.4	Data Channel Protection	13
2.4.1	Copy Control Information.....	13
2.4.2	Copy Control Information Definition.....	13
2.4.3	CCI Authentication Protocol.....	13
2.4.4	Rules for EMI value vs. CA Encryption (to POD) vs. CP encryption (from POD)	13
2.5	Identifying Fraudulent Devices and Disabling of Services	14
2.6	POD Module Data Channel Transactions.....	14
2.6.1	NRSS Copy Protection Framework Messages.....	14
2.6.2	Opening a Copy Protection Session.....	15
2.6.3	Host Evaluation.....	15
2.6.4	Host Authentication	15
2.6.5	Display Message for Unidirectional Systems	16
2.6.6	Authentication Key Verification.....	16
2.6.7	Key Derivation.....	17
2.6.8	POD Module Synchronization	17
2.7	Operational Parameters.....	17
2.8	Command Interface – Application Layer	18
2.8.1	Copy Protection Framework.....	18

2.8.2	APDU Objects and Parameters	20
3.	HOST AUTHENTICATION MECHANISMS	22
3.1	Protocol Components.....	22
3.1.1	ECC-DSA Certificate.....	22
3.1.2	Device Parameters	22
3.1.3	System Parameters	22
3.1.4	Processing Basics.....	23
3.2	POD/Host Binding and Registration	27
3.2.1	ID Report-back Mechanism	28
3.2.2	Authentication Step 1 – Certificate Verification & DH Key Exchange	28
3.2.3	Authentication Step 2 Authentication Key Verification	29
3.2.4	Authentication Step 3 – Headend Report Back.....	29
3.3	Power-up Re-Authentication	32
3.4	CP Key Refresh.....	32
3.4.1	Key Session Period	32
3.4.2	Key Refresh Period.....	33
3.4.3	CAS Key Refresh	33
3.4.4	Key Refresh Initialization.....	33
3.4.5	Channel Change	33
3.4.6	Two Key Synchronization Mode (Informative).....	34
3.5	POD Operation with Multiple Hosts	35
4.	CRYPTOGRAPHIC FUNCTIONS.....	36
4.1	Authentication Key Generation	36
4.2	Copy Protection Key Generation.....	37
4.2.1	Basic Key Generation Protocol.....	37
4.2.2	POD Module Copy Protection Key	38
4.2.3	Host Copy Protection Key	39
4.3	Copy Protection Key Refresh.....	39
4.4	Diffie-Hellman Key Exchange Algorithm.....	40
4.4.1	Algorithm Overview	40
4.4.2	Algorithm Implementation	41
4.5	SHA-1 Secure Hash Algorithm.....	41
4.6	Random Number Generation.....	42
4.7	Elliptic Curve Digital Signature Algorithm (EC-DSA) [Informative]	42
4.7.1	Elliptic Curve Domain Parameters [Informative].....	42
4.7.2	EC Key Pair Generation [Informative].....	42
4.7.3	ECDSA Signature Generation [Informative]	42
4.7.4	ECDSA Signature Verification [Normative]	43
4.8	DFAST Algorithm.....	43
4.8.1	Algorithm Overview	43
4.8.2	DFAST Characteristics	43

5.	HOST SERVICE REVOCATION MECHANISMS	44
5.1	System Issues	44
5.2	Revocation Circumstances	44
5.3	Fraudulent Host Identification	44
5.4	CAS Revocation & Selective Denial of Services	45
5.4.1	Definition of Revocation	45
5.4.2	Selective Service Denial	45
5.5	The Revocation Process	46
5.6	Implementation in the Headend	46
6.	COPY CONTROL INFORMATION (CCI)	47
6.1	CCI Definition	47
6.1.1	EMI - Digital Copy Control Bits	47
6.1.2	APS - Analog Protection System	47
6.2	Associating CCI with a Service	48
6.3	Conveying CCI from Headend to POD	48
6.4	Conveying CCI from POD to Host	48
6.4.1	OCCI Delivery Instances	48
6.4.2	Authenticated Tunnel Protocol	49
7.	TRANSPORT ENCRYPTION FROM POD TO HOST	50
7.1	MPEG DES Encryption	50
7.2	Transport Processing	50
7.3	Timing of Encryption Transitions	51
7.4	Debug Modes Prohibited	51
7.5	Copy Protection Encryption as a function of CA Encryption and EMI Value	51
8.	HOST, POD, & HEADEND MESSAGING PROTOCOLS	52
8.1	Message Protocol Overview	52
8.2	POD & Host Common Messages	53
8.2.1	Opening a Session	53
8.2.2	Host Capability Evaluation	55
8.2.3	Copy Protection Key Generation	57
8.2.4	Host and POD Synchronization	60
8.3	One-way System IAP Message Protocol	62
8.3.1	Protocol Flow Overview	62
8.3.2	Host Authentication Messages	65
8.3.3	Host Authentication Key Verification Messages	68

8.4	Two-way System POD CPS Message Protocol	70
8.4.1	Protocol Flow Overview	70
8.4.2	Host Authentication Protocol Implementation	71
8.5	CCI Simple Authentication Tunnel Protocol (SATP) Messages	73
APPENDIX A. LUHN CHECK DIGIT		77
APPENDIX B. APPLYING CPKEY TO DES ENGINE		78

List of Figures

Figure 1	POD Copy Protection System Overview	6
Figure 1-A.	POD-Host CP Operation.....	26
Figure 1-B.	Headend CP Operations.....	27
Figure 1-C,	Copy Protection System Failure Notification Message	29
Figure 1-D	CP Session Flow Chart	34
Figure 2	Overall POD Copy Protection	36
Figure 3	Diffie-Hellman Key Agreement Protocol between POD and Host	40
Figure 7	CCI Delivery Sequence	48
Figure 5	Copy Protection Message Protocol Overview	52
Figure 6	One way System POD CPS Message Protocol Overview	63
Figure 7	Two-way System POD CPS Message Protocol Overview.....	70

List of Tables

Table 1	Key Derivation Sequence: Parameter and Key Sizes	11
Table 2	CCI Values	13
Table 3	CP Encryption Dependency on CA Encryption and EMI value	14
Table 4	Copy Protection Resource Class	14
Table 5	POD CP APDU Tags	15
Table 6	Opening a Session.....	15
Table 7	Host Capability Evaluation	15
Table 8	Host Certificate Retrieval and Diffie-Hellman Key Exchange.....	16
Table 9	(removed)	16
Table 10	Authentication Key Verification	16
Table 11	Key Derivation Message	17
Table 12	(removed)	17
Table 13	Operational Parameter Values for POD	18
Table 14	CP transport_scrambling_control_field Descriptor Values	21
Table 15	Length of Device Parameters in the Host Authentication IAP	22
Table 16	Length of System Parameters	23
Table 17	Length of Keys and Parameters Used in the Key Generation	39
Table 18	CRL Based Host Service Revocation.....	46
Table 19	CCI Bit Assignments.....	47
Table 20	EMI Values and Content	47
Table 21	CP Encryption of Content based on CA Encryption State and EMI Value	51
Table 22	Copy Protection Open Session Information.....	53

Table 23	Open_Session_Request() Message Syntax	54
Table 24	(removed – see Table 4)	54
Table 25	Open_Session_Response() Message Syntax	55
Table 26	(removed)	55
Table 27	Host CP Support Capability Evaluation Messages.....	55
Table 28	CP_open_req() Message Syntax	56
Table 29	CP_open_cnf() Message Syntax	56
Table 30	CP_system_id_bitmask Values	57
Table 31	CP_data in the Transmission Key Generation Messages	57
Table 32	CP_data_req() Message Syntax In the Key Generation Messages.....	58
Table 33	Datatype_ID and Datatype_length Values.....	59
Table 34	CP_system_id Values	60
Table 35	CP_data_cnf() Message Syntax In the Key Generation Messages.....	60
Table 36	Host and POD Module Synchronization Messages.....	61
Table 37	CP_sync_req() Message Syntax.....	61
Table 38	CP_sync_cnf() Message Syntax.....	61
Table 39	Status_field Value	62
Table 40	(removed)	62
Table 41	(removed)	62
Table 42	(removed)	62
Table 43	(removed)	62
Table 44	One-way System POD CPSP Message Reference Sections.....	64
Table 45	IAP Host Authentication Messages	66
Table 46	CP_data_req in the IAP Host Authentication Request Message	66
Table 47	CP_data_cnf in the POD CPS Host Authentication Response Message	67
Table 48	Host Authentication Key Verification Messages	68
Table 49	CP_data_req in the Authentication Key Verification Request Message	68
Table 50	CP_data_cnf in the Authentication Key Verification Response Message.....	69
Table 51	Two-way System POD CPS Message Reference Sections	71
Table 52	CCI Simple Authentication Tunnel Protocol Messages.....	73
Table 53	CP_data_req() Message Syntax in SATP Key Generation	74
Table 54	CP_data_cnf() Message Syntax in CCI SATP Key Generation.....	74
Table 55	CP_data_req() Message Syntax in CCI SATP Transmission	75
Table 56	CP_data_cnf() Message Syntax in CCI SATP Transmission.....	76

1. Introduction

Superseded

This copy protection specification defines the means to protect high value content on the interface between the Point of Deployment (POD) Removable Security Module and the OpenCable Host device (Host). The overall content protection system is discussed at a high level in chapter 2, followed by a detailed treatment in chapters 3 through 8. Readers seeking a high-level perspective should read chapters 1 and 2, while those seeking implementation details will find them starting in chapter 3.

Content, which is delivered with ‘copying permitted’, e.g., free access off-air broadcast content, is not copy protected and the means described in this specification do not apply to it. Such content may be encrypted from headend to POD but will be delivered in the clear on the POD Host Interface.

‘Copying permitted’ content may be delivered either in the clear (unencrypted) or encrypted from the headend to the POD and will be output in the clear from the POD to Host with Copy Control Information (CCI) set to zero.

The objective of copy protection is to secure protected content against unauthorized¹ access throughout the entire delivery chain from source to display. Program providers have deployed means to secure content from source to the cable headend and cable systems have similarly deployed secure systems from headend to home. Cable set-tops use copy protection technology to protect content on the analog and digital outputs to consumer displays.

With the introduction of the POD Module, cable CAS security will terminate in the POD. A means is needed to prevent unauthorized access on the POD↔Host interface. This document specifies such a means. Basically, the POD Module shall decrypt services under control of the headend and shall re-encrypt content for the purpose of copy protection across the interface between the POD Module and Host device.²

The OpenCable Set-top and POD Module Specifications shall provide for the following:

- a) Device authentication. Prior to binding of the POD Module and Host, the Host is authenticated using its “Restricted Authentication Device Certificate” (as defined in the “5C” DTLA Specification, Vol. 1, Sec. 5.22) which contains a unique ID (*Host_ID*).
- b) The POD↔Host interface is protected using:

¹ “Authorized” is used to here-in to mean permitted by the policy and procedures of the cable operator, CA system, and POD-CP system design. Content, e.g., video services, will only be provided to the OC Host as authorized by these systems. Typically the POD is authorized by the CA system to decrypt services, the POD authorizes the Host by delivering either clear or CP encrypted content. For CA encrypted content not authorized for delivery to the Host the POD will not perform the CA decryption step and will pass through the typically unusable CA encrypted content.

² The copy protection mechanism described in this document does not include a “fingerprint” or “watermarking” mechanism to uniquely identify the source of a specific copy.

- i) Integer field, 1024 bit Diffie-Hellman key exchange with DFAST intellectual property incorporated into the key exchange process.³
 - ii) Encryption of protected MPEG data across the interface, using DES encryption.
 - iii) Authentication of Copy Control Information (CCI) during transmission from POD to Host. The POD will receive the CCI through an authenticated CA System message, and transfer it to the Host using a specified authentication protocol.
- c) Copy Protection on Host device outputs. The digital Host device will support Macrovision copy protection on standard-definition analog outputs⁴ and will use “5C” DTLA copy protection on digital 1394 outputs (per SCTE Standard DVS-194) when these outputs are present. Digital Host devices with other outputs will be granted a license to implement OpenCable POD Module Interface Technology only if they can satisfactorily protect copy protected material.
- d) Revocation of selected services. The cable operator’s Conditional Access System (CAS) will maintain a list of validated Host devices. When a Host is determined to be fraudulent the CAS will selectively deny the appropriate encrypted services to the POD/Host. The denial of service may apply to all protected content or to specific content as determined by the CAS. For example, if properly enabled, the CAS may perform the following:
- i) Cut off service to a single channel, such as “HBO”. This could be done through an EMM, which would selectively deny service based on a Content Provider’s concerns about copy protection.
 - ii) Cut off service on a program-by-program basis. This might be done through an ECM, which would prevent descrambling based on a flag. It addresses the Content Provider’s concern about a particular program being sent to a fraudulent or non-validated Host.
 - iii) When a Host cannot be validated, e.g., it is lacking a valid certificate, the CA System will deny all copy-protected services to the POD/Host.
- e) Service restoration. The CAS will have the ability to deliver either a targeted or a broadcast message that authorizes the restoration of services to a POD that is mated to a Host previously identified as fraudulent but then cleared or revalidated.

1.1 Purpose

This document is intended to identify the implementation steps and corresponding message protocol details for the POD Module and Host. Detailed descriptions of message syntax, message processing, and cryptographic mechanisms used by the protocols are included. Chapter 2 is a high level overview, while chapters 3 through 8 are implementation oriented.

³ CableLabs has made arrangements to license the DFAST technology (U.S. Patent number 4,860,353 and related know-how) necessary to implement the OpenCable POD Module Interface Technology, to any interested party under fair and reasonable terms. For licensing information, refer to the OpenCable website <www.opencable.com> or contact CableLabs at 303/661-9100. OpenCable POD Module Interface Technology is defined as all relevant POD Module interface specifications in addition to the DFAST intellectual property.

⁴ Standard-definition analog is defined as a composite analog signal no greater than 525i resolution.

Messages exchanged to and from the Cable headend are also included to illustrate the processing steps required in the overall system. This document does not specify the full details of all message delivery paths, processing steps and syntax to and from the headend, as these will typically be private mechanisms that differ across CA and cable systems.

1.2 Revision History

Rev	Number	Description	Date
2	DVS213	Copy Protection for POD Module Interface	6/1/99
Oct 8	IS-POD-CP-WD01-991008	Major revisions based on MSO input.	10/8/99
WDO2	IS-POD-CP-WD02-991027	Extensive text revision. Changed POD_ID and Host_ID to 64 bits. Added headend initiated CP Key refresh	10/27/99
WD02.1	Editor's draft	Added periodic CP Key refresh, CCI delivery authentication, Host_ID 40 bits, many editorial changes,	12/6/99
INT01	IS-POD-CP-INT01-000107	Editorial, table of POD input/output encryption vs. CCI, define POD 'Passthrough' state,	1/7/00
INT02	IS-POD-CP-INT02-000410	Incorporate ECN numbers: 32, 61, 62, 63, 64, 66, 70, 71, and 89.	4/10/00
INT03	IS-POD-CP-INT03-000714	ECN 31 to 32 above, new ECN's 65, 67, 68, 73 to 77, 113, 114a, 116, 121, 122, 140. Errors corrected in APDU numerical values of chapter 8.	7/14/00
INT04	IS-CP-INT04-010212	ECN-00097a, ECN-00112, ECN-00141, ECN-000148, ECN-000154, ECN-00176, ECN-00194, ECN-00202	3/7/01
INT05	IS-CP-INT05-010515	ECN-00196	4/18/01

1.3 References

The following documents are applicable to this proposal:

- NRSS, EIA-679 Part B.
- 5C IEEE 1394 Proposal, rev. 1.0, "5C Digital Transmission Content Protection Specification", Volume I, February 18, 1999.
- Digital Transmission Protection License Agreement, see <http://www.dtcp.com>.

- DFAST encryption technology (U.S. Patent number 4,860,353 and related know-how) is licensed from CableLabs as part of the OpenCable POD Module Interface Technology and described in CableLabs' license materials. This technology shall be licensed to any interested party under fair and reasonable terms. For licensing information, refer to the OpenCable website <www.opencable.com> or contact CableLabs at 1(303)661-9100.
- FIPS PUB 186-1, "Digital Signature Standard" available at <http://www.itl.nist.gov/fipspubs/>
- FIPS PUB 140-1 "Security Requirements for Cryptographic Modules"
- ANSI X9.62 and IEEE P1363 Elliptic Curve Digital Signature Algorithm standards

1.4 Acronyms and Abbreviations

The following acronyms and abbreviations are used within this proposal:

5C-POD	The DTCP Certificate specific to the POD↔Host interface.
APDU	Application Protocol Data Unit (APDU)
AK	Authentication Key, calculated by both POD and Host as a simple means of confirming that the Host is authentic. Host ECC-DSA certificate confirmation is also used to confirm an authentic Host.
CCI	Copy Control Information.
CAS	Conditional Access System – secures delivery of cable services on the cable distribution system.
CP	Copy Protection
CPK	The Copy Protection Key, (K_s_{dfast}) derived between the POD and Host, and used in the POD to encrypt MPEG packets sent to the POD.
CPS	Copy Protection System
CRL	Certificate Revocation List: the means of reporting bad Host ID's for distribution to cable headends for deauthorization via the CAS. The term "CRL" is a generalized standard term, not specific to 1394-5C. DTLA conveys the CRL to users in a "System Renewability Message", or SRM.
CSR	Customer Service Representative – a telephone response employee
DH	Diffie Hellman, a public key agreement protocol based on the intractability of taking discrete logarithms over the integer field.
DSA	The US standard Digital Signature Standard algorithm for signing and verifying signatures and certificates.
DTLA	Digital Transmission License Authority
DTCP	Digital Transmission Copy Protection
ECC-DSA	The elliptic curve implementation of DSA
EMI	Encryption Mode Indicator

IAS	Interface Authentication System
POD CPS	POD Copy Protection System
POD CPSP	POD Copy Protection System Protocol
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.

1.5 Copy Protection System Components

This copy protection system (CPS) includes:

- The cable navigation device, or Host
- The POD Module
- The Cable Headend (since revocation of selected services of compromised Hosts occurs there)
- The Digital Transmission License Authority who is responsible for generating device certificates and assigning device IDs.

1.6 Implementation Outline

The POD Copy Protection System (POD CPS) focuses primarily on POD↔Host interface security and is supported by the POD Copy Protection System Protocol (POD CPSP). The POD CPS implementation is outlined as follows:

- The POD Copy Protection System message protocol, which defines exchange of randomized public keys across the POD-Host interface using the integer field Diffie-Hellman protocol;
- A certificate with a DTLA 5C Elliptic Curve Digital Signature Algorithm (ECC-DSA) signature and certificate, where DTLA is assumed to produce this certificate;
- Verification of the ECC-DSA Host certificate by the POD;
- Host ID and POD Module ID headend report-back mechanism.
- Certificate Revocation List (CRL) delivered and updated in the Headend to revoke selected services of a compromised Host.

The POD CPS implementation is shown in Figure 1.

Copy Protection System Components & Algorithms

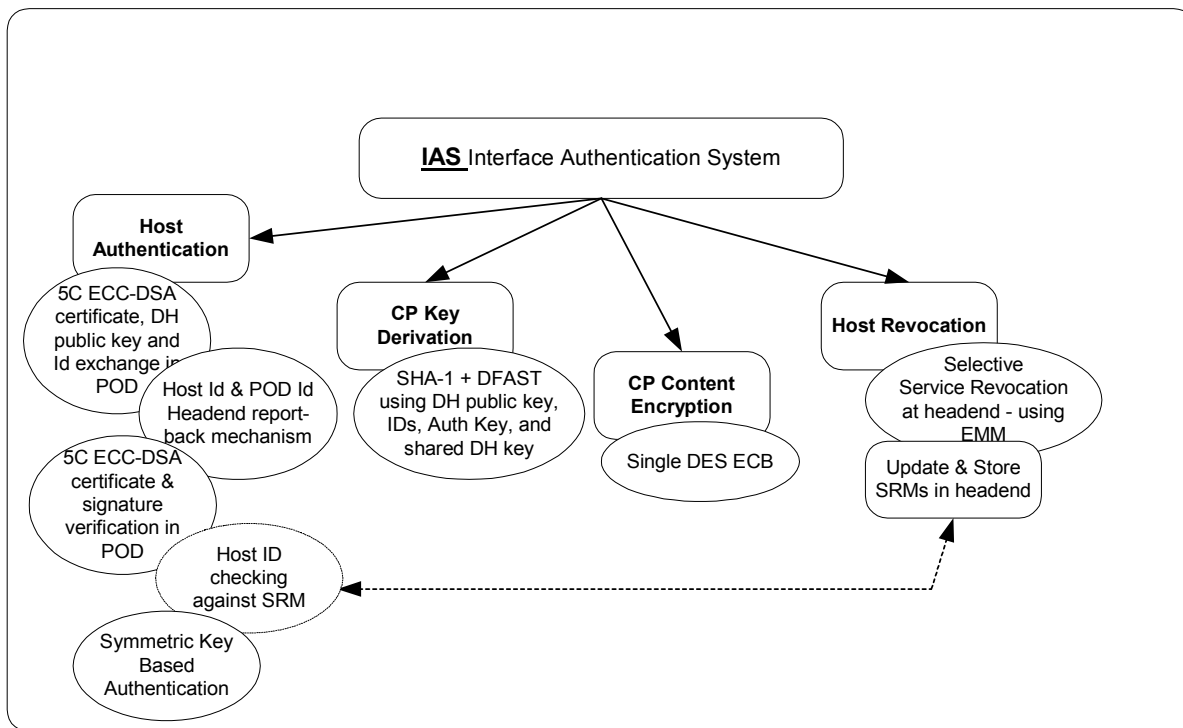


Figure 1 POD Copy Protection System Overview

1.7 Document Links [Informative] ⁵

This document is intended to supplement the functionality of the POD Module interface described in IS-POD-131-INT05-010307 which defines how copy protection fits into the overall POD Module interface specification. This document defines the details of how that functionality are implemented.

Much of the content for this document is based on the Copy Protection Framework described in Section 8.9 of EIA-679B part B. However, at the time of this writing, a final version of EIA-679B part B had not been approved by the EIA organization. As a result, extracts from the most current available version of EIA-679B part B, Section 8.9 have been included in this document as sections 2.6, 2.7, and 2.8.

⁵ Unless labeled “Informative” in the section title, all sections in this document are Normative and required.

2. System Overview

2.1 NRSS Copy Protection Framework

This document is substantially compliant with draft version EIA-679 part B, National Renewable Security Standard, Part B, Section 8.9, Copy Protection Framework. The NRSS Copy Protection Framework includes:

- POD/Host Binding
- DES-ECB Scrambling System (Inband Channel)
- MPEG2 System Layer (Inband 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.

- Host authentication
- Host selective service revocation

Under the NRSS Copy Protection Framework, the POD Module shall check the ability of the Host to support OpenCable™ Content Protection by checking availability of the Copy Protection Resource as defined in DVS-131 and verifying the authenticity of the 5C-POD Device Certificate. If the Host is not compliant, the POD Module will not descramble any of the content-protected materials and will pass the bitstream back to the Host the same as it was received in "passthrough" mode.

2.2 Host Authentication

The OpenCable™ Content Protection System requires the Host device to be authenticated prior to the POD descrambling any of the content-protected material. The POD shall send a request to the Host device for the 5C-POD Device Certificate. The Host will return the 5C-POD Device Certificate to the POD. This authentication is based on:

- POD being able to verify the signature of the 5C-POD device certificate that contains Host_ID; and
- POD being able to prove it holds the same secret key as the Host does, so the same authentication key and Diffie-Hellman key can be derived in both POD and Host.
- The Host ID extracted from the certificate is not included in the CRL, as checked in the Headend.

An Authentication Key (AK) is generated when the POD and Host are first connected. The key values resulting in both POD and Host are then stored in non-volatile memory, and are used to generate the Copy Protection Key later in the key derivation step.

The headend will validate the *Host_ID* and *POD_ID* and verify that services associated with that *Host_ID* have not been revoked. If valid, the headend sends the *Host_ID* back to the POD via an authenticated message as *Validated_Host_ID* through the operator's CAS.

Additionally, the POD will validate the authenticity of the 5C-POD Device Certificate using the License Administrator's public verification key that is delivered by the CAS to the POD in an authenticated manner. If the *Validated_Host_ID* value is the same as the *Host_ID* in the authenticated 5C-POD Device Certificate, the 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 report back transmissions to the headend, via either RDC on the cable plant or via telco modem to the Host, the POD will send *POD_ID* and *Host_ID* on that channel without the need for manual reporting.

2.2.2 Manual Return Authentication

For one-way cable systems, Uni-Directional Hosts, or any system without an internal report back mechanism, reporting of the authentication ID's must be accomplished manually. The POD module shall always include a diagnostic menu item in the application info APDU which allows for a message which contains the Host ID and POD ID to be displayed to the user (see below). 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.

The POD module shall send a request to the Host device for its 5C-POD Device Certificate. If the POD module has not mated with this Host and is either in a unidirectional system or is inserted in a unidirectional Host, the POD module shall then 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 HTML message containing the *Host_ID* and *POD_ID*. This message will contain the *Host_ID* and *POD_ID*, each with a Luhn check digit (described in appendix A) appended, in decimal format (digits 0-9) grouped in three digit sets so that they are easier to read and speak, and can be entered from a touch-tone telephone keypad. Prior to receiving protected content the *Host_ID* and *POD_ID* shall be reported to the service provider, either by telephone or some other method. The end-user or the retailer may perform this service.

The unidirectional message screen shall be displayed only if 1) the message is selected through the menu, 2) mating has not occurred and the user selects a program with copy protection active (*CCI* ≠ 0).

An example of a displayed message follows, where both the POD and Host ID's are presumed to be 40 bits in length (13 decimal digits each plus a 1 digit checksum):

In order to start service for this device,
please contact customer service at
1-800-555-8888

POD_ID = xx-xxx-xxx-xxx-xxx
Host_ID = xx-xxx-xxx-xxx-xxx

Thank You

2.2.3 Manual Return Authentication – Error and Other Conditions

The Host has the capability of requesting the unidirectional message screen via the menu through the application info resource. There are three conditions in which the unidirectional screen described in section 2.2. is not valid:

- The Host and POD module have not completed their mating.
- The Host has an invalid certificate.
- The Host is bidirectional and the POD module has established a reverse path.

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.

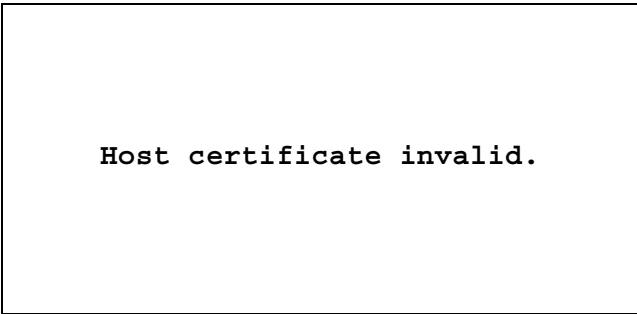
2.2.3.1 Incomplete Mating

In the event that the Host and POD module have not completed phase one of mating authorization (validation of the Host certificate) at the time the Host requests the copy protection message screen, the POD module shall display the following message or its equivalent:

Information not available.

2.2.3.2 Invalid Certificate

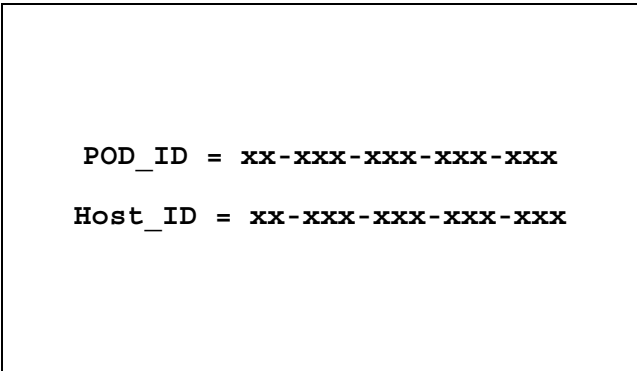
In the event that the Host supplies an invalid certificate to the POD module, then if the Host requests the copy protection message screen, the POD module shall display the following message or its equivalent:



Host certificate invalid.

2.2.3.3 Normal Diagnostic Operation

If the Host requests the copy protection message screen and the Host ID has been received and authenticated, independent of whether the Host or system are unidirectional or bidirectional, then the POD module shall display the following message or its equivalent:



POD_ID = xx-xxx-xxx-xxx-xxx
Host_ID = xx-xxx-xxx-xxx-xxx

The x's are to be replaced by the values of the POD ID, Host ID, and their respective Luhn checksums.

2.3 Key Exchange and Interface Encryption

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 to the manufacturer by CableLabs when the license to implement OpenCable POD Module Interface Technology is granted. PODs and Hosts also contain private keys (p and h , respectively) and the corresponding public keys ($g^p \bmod n$ and $g^h \bmod n$). The private keys p and h are pseudorandom integers⁶ that are 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).

⁶ The pseudorandom integer generator used to generate p and h shall be compliant with the SHA-1 based algorithm described in FIPS PUB 186-1, Appendix 3, Section 3.3. Each OpenCable device shall have a uniquely generated seed value that is set in the factory. The seed generator shall comply with the FIPS PUB 140-1 Section 4.11.1 test for randomness.

2.3.1 Setup Phase

If the POD is authorized to do so (e.g., by the headend or by policy), 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-bits 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 (CPK).

2.3.1.1 Authentication Key Setup

Normally an authentication key ($AuthKey_P/AuthKey_H$) only needs to be calculated when the POD is inserted into a Host the first time. At the initial insertion or power on, the Host authentication process is conducted and the authentication key is saved in non-volatile memory.

There are two circumstances in which the authentication key must be re-calculated. The first is when the authentication key provided by the Host at power on does not match the previously stored authentication key. The second is when re-initialization is commanded by the CA System in the headend (see section 3.2.3.)

The authentication key generation process is detailed in section 3.2.4.

2.3.1.2 Long Term Diffie-Hellman Shared Key Value

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

2.3.2 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. If the CCI has been altered before being transmitted to the Host, the Host will be able to detect such alteration and not use the suspect CCI. The POD will verify the mated Host has received the correct CCI through an authenticated acknowledgement message.

2.3.3 Key Derivation Phase

After a POD Module and Host have been bound, the Copy Protection (CP) key shall be derived. CP encryption key is calculated based on the long-term keys, Authentication and DH keys, and the random number exchanged for key generation. Details are discussed in chapters 3 and 4.

Table 1 Key Derivation Sequence: Parameter and Key Sizes

Parameter/Key	Size (bits)
$AuthKey_P, AuthKey_H$ (authentication keys)	160
$DHKey$ (shared DH secret)	1024
N_{Host}, N_{module} (Host and POD nonces)	64
CP_Key (Copy Protection Key)	56

2.3.4 Multiple Host Support

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 not compliant with this specification. Each POD must bind to only one Host. If multiple Host capability is determined to be important it may be added to a future extension of this specification but is specifically proscribed here.

2.3.5 Interface Encryption

The POD uses encryption to protect the transmission of copy-protected content across its interface with the Host. The POD re-encrypts the content it has decrypted 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.

2.3.5.1 Scrambling

Content passing the POD interface may exist in any of the following three formats:

- Cleartext
- Pass through
- Rescramble

In order for the Host to find out which format is used for a particular content, it shall send a CA_PMT message with ca_pmt_cmd_id parameter set to “ok_descrambling” to the module. If the POD replies with a CA_PMT_REPLY, the content is Passed Through, otherwise it can be either Cleartext or Rescrambled. The “Scramble” bit in the MPEG TS header will allow differentiating the last two options.

The CA System shall specify which format shall be used for each program.

2.3.5.1.1 Key Derivation Sequence

After a POD and Host have been bound, the CP key shall be derived. See chapters 3 and 4. After the first derivation ends and CP_Sync_Cnf has been sent, the POD may begin scrambling data with the key.

2.3.5.1.2 Key Application for In-Band Channel

The Copy Protection Key shall be applied across all rescrambled packets. Only a single copy-restricted MPEG program will be delivered from the POD to the Host.

2.3.5.1.3 Content (In-Band Channel) Encryption

DES⁷ scrambling shall be performed on the payload portion of the packets in the transport stream using the electronic code book version of DES (DES-ECB⁸). DES-ECB scrambles data in blocks that are 8 bytes long, however there may be a short block (e.g. less than 8 bytes) remaining at the end of

⁷ The DES cipher is described in FIPS PUB 46-2.

⁸ The different modes in which the DES cipher can be used (including DES-ECB) are described in FIPS PUB 81.

an encrypted transport packet. The ECB DES shall be applied block-by-block, starting at the beginning of the transport packet body. Any short block will therefore occur at the end of the transport packet. The transport packet header and adaptation field (if any) shall not be scrambled. The short block shall not be scrambled. See chapter 7 for more information.

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 shall be authenticated, as described in the CCI chapter.

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. If the CCI or has been altered before being transmitted to the Host, the Host will be able to detect such alteration and not use the suspect CCI.

2.4.2 Copy Control Information Definition

This one-byte field contains information that the Host shall use to control copying of content received from the POD; see Table 2 for CCI value definitions. Four bits are defined. Two EMI bits control digital copy permissions. Two APS bits control copy protection encoding of standard analog output video. The POD shall be downloadable to support extended CCI processing at a future time.

Table 2 CCI Values

CCI Bit	7	6	5	4	3	2	1	0
Definition	reserved	reserved	reserved	reserved	APS1	APS0	EMI1	EMI0

2.4.3 CCI Authentication Protocol

CCI information shall be authenticated when it is traveled across the interface. The message authentication process has the following steps:

1. POD generates an 8 byte random number, *CCI_N_module*, and sends it to the Host.
2. Host generates an 8 byte random number, *CCI_N_Host*, and sends it to the POD Module.
3. POD sends an authenticated CCI message to the Host.
4. The Host verifies the message and replies with an authenticated acknowledgement.

2.4.4 Rules for EMI value vs. CA Encryption (to POD) vs. CP encryption (from POD)

1. CP encryption shall be applied only to copy-protected content, EMI not equal to 00.
2. All copy-protected content shall be CA encrypted to the POD.

3. Non-copy-protected content, with EMI = 00, may be delivered to the POD CA encrypted or as cleartext (un-encrypted).

Table 3 CP Encryption Dependency on CA Encryption and EMI value

If: CA Encrypted	If: Copy Protected? (EMI not 00)	Then: POD CP Encrypts Output?
No	No	No
Yes	No	No
No	Yes	Disallowed State
Yes	Yes	Yes

2.5 Identifying Fraudulent Devices and Disabling of Services

The OpenCable™ Content Protection system requires the POD Module to retrieve the OpenCable™ Content Protection Operation Parameter “*CP_Host_certificate*” from the Host. The POD verifies this certificate when a POD Module is mated with a Host. When two-way capability is available, the POD reports the *Host_ID* upstream to the headend, and the Headend confirms the *Host_ID* back downstream to the POD.

When two-way capability is not available, the *Host_ID* and *POD_ID* are reported to the headend by the end-user via telephone (touch-tone or voice). When it has been determined by the MSO that a fraudulent Host device should no longer receive a service or group of services, the MSO conditional access system will issue commands to the POD denying the appropriate services. Restoration of services is also executed by the conditional access system. This process of CA System-based revocation is known as *Selective Denial*.

Revocation of selective services for fraudulent Host devices is detailed in chapter 5.

2.6 POD Module Data Channel Transactions

2.6.1 NRSS Copy Protection Framework Messages

The OpenCable™ Content Protection system refers to the NRSS Copy Protection Framework messages and resources to exchange data across the Data Channel. These messages have been used wherever possible in this POD protocol.

Table 4 Copy Protection Resource Class

Resource	Class	Type	Version	Identifier
Copy Protection	176	2	1	00B00081

Table 5 POD CP APDU Tags

APDU Tag	Tag Value (Hex)	Resource	Direction Host ↔POD
CP_open_req()	9F9000	CopyProtection	←
CP_open_cnf()	9F9001	CopyProtection	→
CP_data_req()	9F9002	CopyProtection	←
CP_data_cnf()	9F9003	CopyProtection	→
CP_sync_req()	9F9004	CopyProtection	←
CP_sync_cnf()	9F9005	CopyProtection	→

2.6.2 Opening a Copy Protection Session

The following uses parameters already defined within EIA-679-B for opening a session to the Copy Protection Resource in the Host. This would typically be granted right away.

Table 6 Opening a Session

#	Action	Object
1	The POD Module requests a session of the Copy Protection resource to be opened	Open_Session_Request()
2	The Host replies with a session status. If successful, a session number is assigned	Open_Session_Response()

2.6.3 Host Evaluation

The NRSS Copy Protection Framework requires the POD Module to check the Host's ability to support the OpenCable™ Content Protection algorithm, when the POD Module is powered on and before starting the Key Exchange process.

Table 7 Host Capability Evaluation

#	Action	Object
3	POD Module queries Host's ability to support OpenCable Copy Protection Systems	CP_open_req()
4	Host replies to POD Module	CP_open_cnf()

If the Host's answer includes the support of *CP_system_id*=2 then the POD Module shall start the Key Exchange process.

2.6.4 Host Authentication

The OpenCable™ Content Protection mechanism requires the POD to retrieve the Host Certificate Data to initiate the authentication procedure and to also start the Diffie-Hellman Key Exchange process.

Table 8 Host Certificate Retrieval and Diffie-Hellman Key Exchange

#	Action	Object
5	The POD Module sends its ID and DH public key to the Host and requests that the Host deliver its Host Certificate Data and DH public key (CP_system_id = 2, send datatype_id's = 6,14, request datatype_id's = 15, 13)	CP_data_req()
6	The Host replies to the POD Module	CP_data_cnf()

After this exchange, both the POD Module and the Host come up with the authentication key, *AuthKey_P* and *AuthKey_H*, using the shared secret value SSK.

2.6.5 Display Message for Unidirectional Systems

The OpenCable™ Content Protection mechanism requires support for Host authentication via subscriber action in a unidirectional system. The MMI resource in the Host will be used to display the messages described in section 2.2.2 to request this action from the subscriber. When the cable system, POD, and Host operate in bi-directional mode the display of these messages, and the subscriber action they request, are typically not required.

Table 9 (removed)

2.6.6 Authentication Key Verification

The OpenCable™ Content Protection mechanism requires verification of the Authentication Key generated by the Host to be verified against the one generated by the POD.

Table 10 Authentication Key Verification

#	Action	Object
15(7)	The POD Module requests that the Host send its AuthKey (CP_system_id = 2, request datatype_id = 22)	CP_data_req()
16(8)	The Host send its AuthKey to the POD Module	CP_data_cnf()

After receipt of the Host's *AuthKey_H*, the POD module verifies that it matches its *AuthKey_P*. If the AuthKeys match, they calculate a common, secret value, *DHKey_s*, and store it in non-volatile memory. If the AuthKeys do not match, the POD shall reject the Host and refuse to authorize high value services with EMI of 01, 10, or 11.

Note that if a POD module believes it holds a valid AuthKey, it may issue this CP_data_req immediately after steps 3 & 4 above (Host Evaluation) to determine if full authentication is required. If the AuthKey received by the Host matches the one stored by the POD, then the POD may proceed to the Copy Protection Key Derivation steps (section 2.6.7). If the Host's AuthKey does not match the POD's stored AuthKey, then the POD must perform full authentication, beginning with step 5 above (Host Certificate Retrieval and Diffie-Hellman Key Exchange) and continuing with another Authentication Key Verification process.

2.6.7 Key Derivation

The OpenCable™ Content Protection mechanism requires starting the key derivation process after a positive Host Authentication as indicated by the POD receiving the *Validate_Host_ID* from the headend.

Table 11 Key Derivation Message

#	Action	Object
9	The POD Module requests the negotiation of a fresh key. This message contains a random seed <i>N_module</i> generated by the POD Module (CP_system_id = 2, send datatype_id = 12, receive datatype_id = 11)	CP_data_req()
10	The Host replies to the POD Module. The response contains a random seed <i>N_Host</i> generated by the HOST	CP_data_cnf()

At the end of the Key Derivation process, both POD Module and Host hold a new key.

2.6.8 POD Module Synchronization

The POD Module need not be capable of seamlessly changing Copy Protection Keys through a new key exchange and derivation process. In single CP Key operation the POD may switch briefly to cleartext on the MPEG output during key refresh sessions. The interface shall return to encrypted output at the earlier of completion of the key change or one second from the start of the key generation session. In dual key operation the POD will output no unscrambled content during CP_Key generation.

Table 12 (removed)

2.7 Operational Parameters

Table 12 lists various NRSS operational parameters used in POD copy protection and POD CPS. These are the only two parameters used from NRSS in POD copy protection.

Table 13 Operational Parameter Values for POD

Operation Parameter	# of bits	Comments
<i>Cert_Host</i>	88 bytes	This parameter is used to pass the 5C-POD Device Certificate.
<i>CP_system_id_bitmask</i>	32-bit bitmask	CP_system_id is set in the CP_system_id_bitmask when the CP_open_cnf() message is sent. CP_system_id is hardwired to '02' for all OpenCable devices.

2.8 Command Interface – Application Layer

2.8.1 Copy Protection Framework

2.8.1.1 Copy Protection Resource Class

A part of the POD Module is an interface to the copy protection (CP) resource class. (See EIA-679 for further information concerning NRSS copy protection systems.) This resource class provides secure communication over the high speed transport stream interface. It is used to protect (scramble) the content carried by the MPEG2 transport stream after the POD Module has descrambled it. Diffie-Hellman (DH) key exchange mechanism is used to establish a shared key and DES as a local rescrambling function.

The resource class is defined in a generic fashion so that different underlying copy protection technologies can be used in a common way. Resource types are defined within the class to support communication to particular types of devices using the common object set. The model is a simultaneous bi-directional channel (full duplex) over which communication of arbitrary data is possible. The APDUs described in chapter 8 are used by the copy protection functions of the Host and the application.

2.8.1.2 Service Provider [Informative]

Involvement of the service provider, i.e. headend or broadcast station, through the private CA System, shall be used instead of POD processing and control. After obtaining the requisite Host information, the service provider, through the CA System, may selectively de-authorize PODs based on certain criteria, for example, if PODs are found to be releasing copy protected content to unprotected Hosts.

2.8.1.2.1 Means of Reporting ID's to the Headend [Informative]

The POD-CP system requires some means to report Host and POD ID's to the service provider. This may be automatic (i.e. using the RDC, DOCSIS or telephone modem), or may be manual, where the user either recites the ID information from the TV screen to a customer service representative or keys the numbers in using a touch-tone telephone keypad. The same information may be conveyed to the service provider in either the manual case or in the automatic case. Existing NRSS commands may be employed by the POD to cause the data to be displayed.

2.8.1.2.2 Service Provider Processing [Informative]

2.8.1.2.2.1 Checking Validity of Host Information

The service provider may be used to check the validity of Host information being delivered (i.e. the Host certificate) by using the appropriate method required for the copy protection technology used.

2.8.1.2.2.2 Revocation of Selected Services for Hosts

The service provider shall perform revocation of selected services for which the Host is authorized. Using proprietary CA mechanisms, revocation of the Host's services 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 then use proprietary CA mechanisms to deliver a “pass/no pass” on the services of a particular Host device, or to deliver new copy protection parameters to the POD. By involving the service provider in the process, a number of different underlying CP requirements may be accommodated.

2.8.1.3 POD/Host Binding

The POD and the Host may negotiate a long term shared secret using Diffie Hellman key exchange. This key is unique to the particular POD-Host pair, and serves to bind them together as well as to locally encrypt content.

2.8.1.3.1 Discrete Logarithm DH

The Discrete Logarithm DH⁹ method is summarized as follows:

- The POD sends its public key $g^p \text{ mod } n$ to the Host.
- The Host sends its public key $g^h \text{ mod } n$ to the POD
- The Host takes $g^p \text{ mod } n$ from the POD, and forms $(g^p \text{ mod } n)^h \text{ mod } n = g^{ph} \text{ mod } n$.
- The POD takes $g^h \text{ mod } n$ from the Host, and forms $(g^h \text{ mod } n)^p = g^{ph} \text{ mod } n$.

The POD and Host have now negotiated a shared secret $g^{ph} \text{ mod } n$. The most significant 128 bits of this secret value shall be defined as K_s , and shall be passed to DFAST to create the 56 bit Copy Protection Key K_{s_dfast} .

The constants (g and n) necessary for implementing this key exchange protocol shall be established by the licensor of the CP System, CableLabs. The length of the prime modulus n is 1024 bits.

⁹ For a full description see page 513 of “Applied Cryptography” by Bruce Schneier.

2.8.2 APDU Objects and Parameters

Six objects shall be defined: CP_open_req(), CP_open_cnf(), CP_data_req(), CP_data_cnf(), CP_sync_req(), CP_sync_cnf(). See sections 8.2, 8.3, and 8.4.

2.8.2.1

(this section intentionally left blank)

2.8.2.1.1 CP_open

The POD uses CP_open_req() and CP_open_cnf() objects to find out which CP systems are supported by the Host.

2.8.2.1.1.1 CP_open_req()

See section 8.2.2.1 for CP_open_req() syntax.

2.8.2.1.1.2 CP_open_cnf()

See section 8.2.2.2 for CP_open_cnf() syntax, where CP_system_id_bitmask values are defined in section 8.2.2.2.

2.8.2.1.2 CP_data

The POD shall use CP_data_req() and CP_data_cnf() objects to provide and request information to/from the Host in the set-up and key generation phases.

2.8.2.1.2.1 CP_data_req()

See section 8.2.3.1, 8.3.2.1, and 8.3.4.1 for CP_data_req() syntax.

2.8.2.1.2.2 CP_system_id

CP_system_id values shall be as indicated in section 8.2.3.1.

2.8.2.1.2.3 Datatype_ID

Datatype_ID values and sizes shall be as indicated in section 8.2.3.1.

2.8.2.1.2.4 CP_data_cnf()

CP_data_cnf() syntax shall be as defined in section 8.2.3.2, , 8.3.2.2, and 8.3.4.2.

2.8.2.1.3 CP_sync

The POD shall use the CP_sync_req() object to notify the Host about its intention to start to copy protect content. The CP_sync_cnf() object is the answer that confirms Host readiness.

2.8.2.1.3.1 CP_sync_req()

CP_sync_req () syntax shall be as defined in section 8.2.4.1.

2.8.2.1.3.2 CP_sync_cnf()

CP_sync_cnf () syntax shall be as defined in section 8.2.4.2.

2.8.2.1.3.3 Status_field

Status_field shall return the status of the *CP_sync_req()*. If the Host is ready to descramble the incoming stream, then *Status_field* shall be set to 0x00. Otherwise, it shall be set to one of the values indicated in section 8.2.4.2.

2.8.2.2 transport_Scrambling Control Field

The transport_scrambling_control field of the ISO/IEC 13818-1 transport packet transport_scrambling_control field bits and descriptor values shall be as defined in Table 14.

Table 14 CP transport_scrambling_control_field Descriptor Values

Bit values	For Single CP_Key Devices	For Optional Dual Key Support Devices
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

3. Host Authentication Mechanisms

3.1 Protocol Components

Host authentication is based on:

- The POD being able to verify the Host's certificate signature.
- The Host ID is extracted from the Host certificate and reported to the headend, where the CA System must confirm that it is not in the CRLs stored in the headend.
- The Host being able to prove it holds the same secret key as the POD, whereby the same Authentication Key and Diffie-Hellman key can be derived in both POD and Host.

3.1.1 ECC-DSA Certificate

A DTLA 5C Host certificate (*Cert_Host*) is used in the Host authentication process. Each Host certificate is assigned by DTLA, and includes the *Host_ID*, and a DSA signature generated by DTLA.

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 15 Length of Device Parameters in the Host Authentication IAP

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

3.1.3 System Parameters

The following system parameters are used in this authentication protocol:

- $Host_ID$: Host ID assigned by DTLA as part of the 5C certificate.
- POD_ID : POD Module ID assigned by POD vendor (in cooperation with the CA System vendor).

- EC-DSA System Parameters: supplied by DTLA, see Table below.
- SSK: The POD and Host receive this system parameter from Cablelabs.

Table 16 Length of System Parameters

Key or Variable	Size (bits)	Source of Parameter
Host_ID	40 bits	Host Manufacturer
POD_ID	64 bits	POD Manufacturer
SSK	128 bits	CableLabs
Diffie-Hellman prime (DH_p)	1024 bits	CableLabs
Diffie-Hellman base (DH_g)	1024 bits	CableLabs
ECC base point G	320 bits	DTLA
Order of base point (r)	160 bits	DTLA
ECC Coefficient of curve polynomial (a, b)	160 bits (each)	DTLA
Prime number (p) of finite field GF(p) (field size)	160 bits	DTLA
ECC-DSA public key y	320 bits	DTLA

3.1.4 Processing Basics

The POD CPS comprises the following basic steps:

- 1) The Host shall report copy protection as a resource during the profile inquiry process (IS-POD-131-INT03-000714). 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 CCI 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 (ECR-161). Failure to respond to any request within 5 seconds constitutes a failure of the Host and will cause the POD module to set the IIR flag.
- 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 AuthKeyH. (ECN-00141)
- 7) The Host shall respond with its AuthKeyH, 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 AuthKeyH.

- 8) The POD module shall compare the received AuthKeyH with its stored AuthKeyP. If the authentication keys match, then the certification verification is considered valid, the DH Key and authentication keys are preserved, and only the Copy Protection Key is regenerated (ECN-00141). Continue with step 14. If the authentications keys do not match, then the POD and Host will continue with the certification validation.
- 9) The POD module shall send its ID and newly generated DH public key and request that the Host deliver its Host Certificate Data (CertH) and DH public key (DH_pubKeyH) (ECN-00141).
- 10) The Host shall reply to the POD module request with its Host Certificate Data and newly generated DH public key.
- 11) The POD module shall verify the CertH 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) After this exchange, both the POD module and the Host come up with the authentication key, AuthKeyP and AuthKeyH, using the shared secret value SSK.
- 13) The POD module shall verify that AuthKeyP is equal to AuthKeyH. If they are not equal, then 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 shared secret DH key (DHKey) into non-volatile memory. The derived authentication keys shall also be stored into non-volatile memory.
- 14) The following depends upon what path is available to report device ID's 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 ID information to the cable headend via an upstream OOB private CA message or via telco modem. The POD module also stores the Host ID so it can be compared with the ID received back from the headend. If this is entered from step 8, then knowledge of this transmission shall be stored in non-volatile memory and the POD shall not retransmit if it has already done so. This is to prevent the system network from being overloaded.
 - b) In systems in which no automated return path is available, the POD module sends a display message as defined in sections 2.2.2 and 3.2.1.
- 15) 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 CAS re-sends the Host ID and POD ID back to the POD module in a private CA System Host_ID validation message.
 - a) This *Host_ID* validation message may be sent back to the POD module substantially later in time than when the *Host_ID* is received from the POD module. Until the POD module receives this message, the POD module shall restrict its authorized services only to those services with a CCI value of 00.
 - b) Asynchronously and without having received the *Host_ID*, the CA system also sends an EMM to the POD module to authorize appropriate services. Some of these services may have CCI values of 01, 10, or 11. The POD module shall not authorize services with these CCI values until the *Host_ID* validation message has been received, even if these services are authorized in an EMM.
 - c) The POD module compares the received Host_ID from the *Host_ID* validation message with the Host_ID extracted from the certificate. If they match, the POD module will allow CA decryption of high value content with CCI 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 CCI values of 00 only.

- 16) If the headend CA systems receives a new revocation list, it shall examine all previously reported Host_ID's and if there are any matches, it shall notify the cable operator.
- 17) 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 the above steps:

POD-Host CP Operation

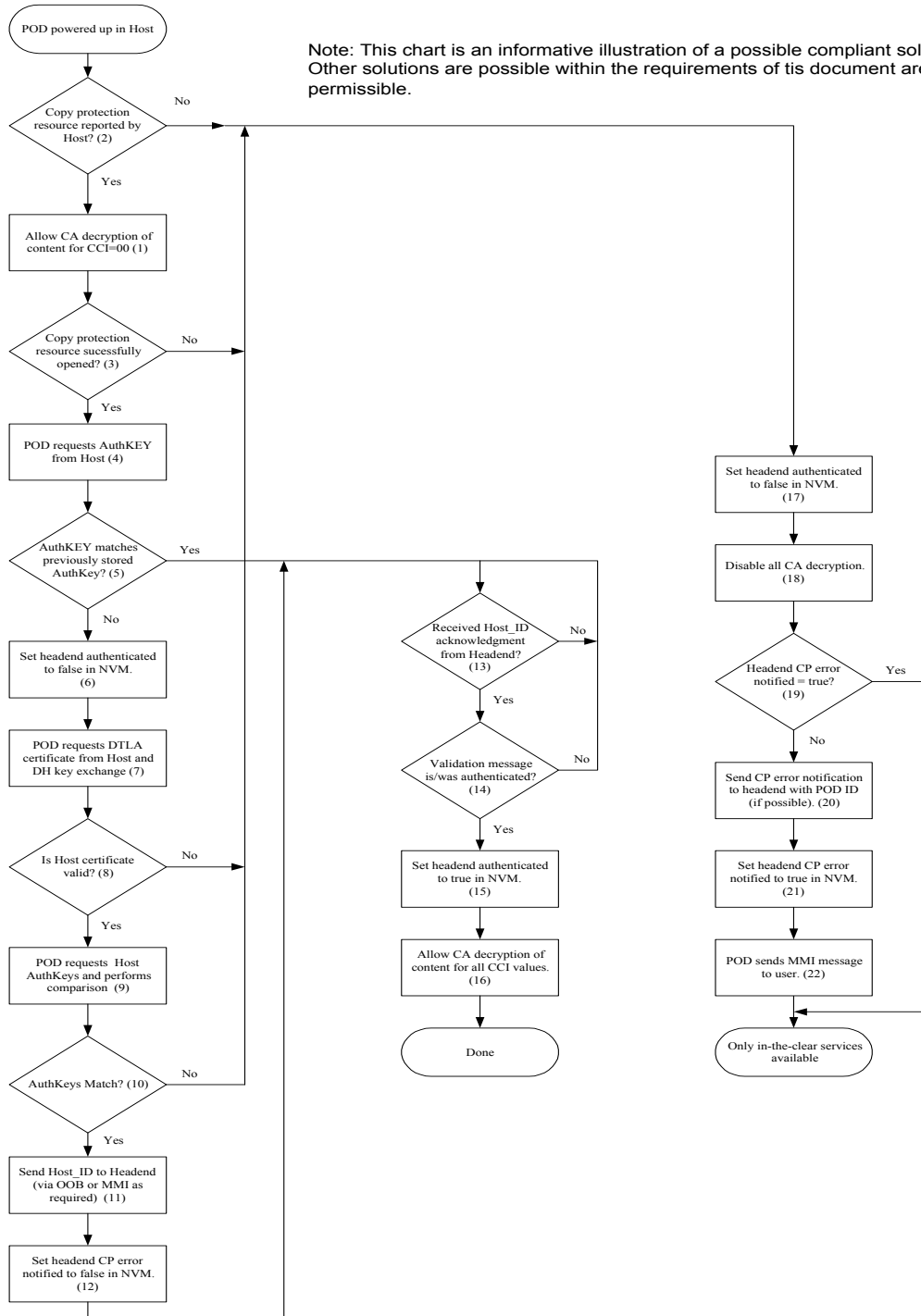


Figure 1-A. POD-Host CP Operation.

Headend CP Operations

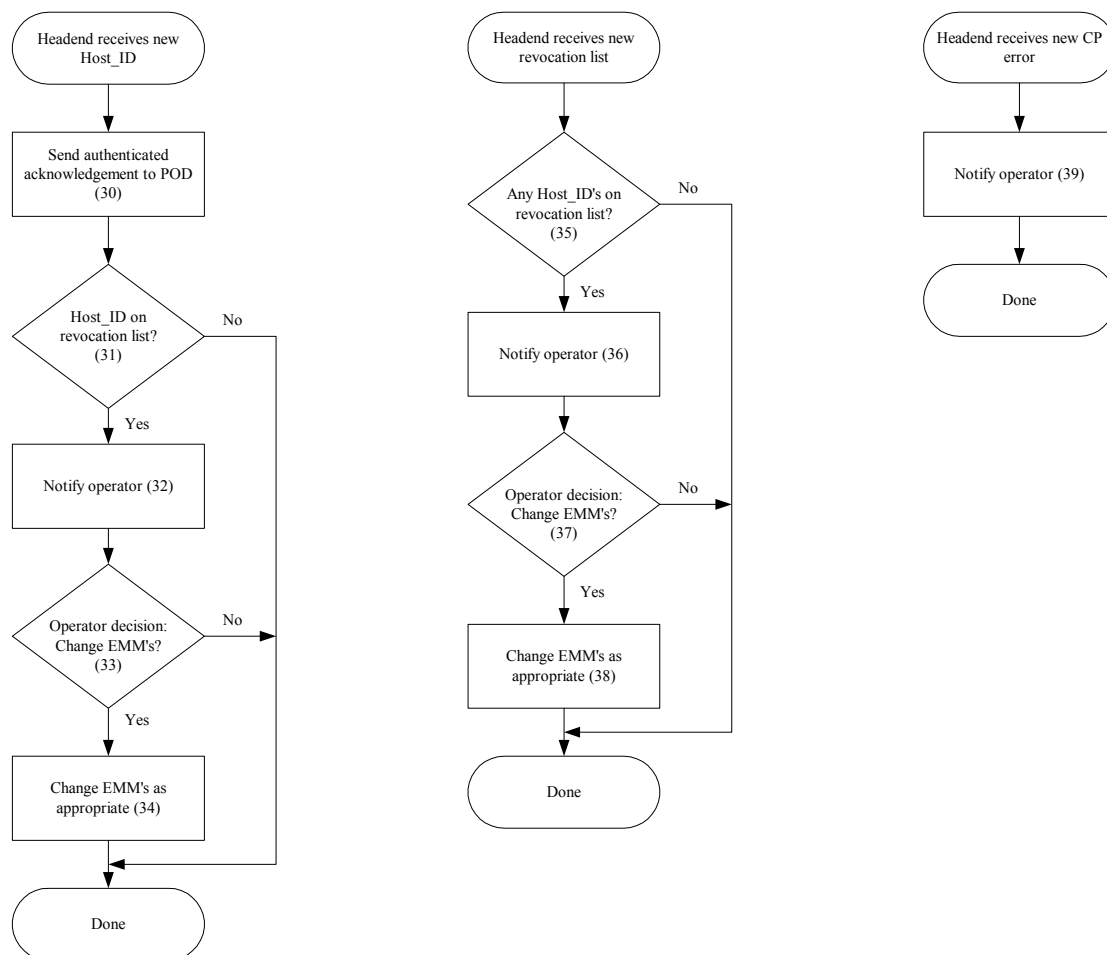


Figure 1-B. Headend CP Operations

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 validation
- Derived Authentication Key setup
- Host ID and POD ID report-back to Headend and Headend validation

The Certificate Revocation List (CRL) is used in the headend (not in the POD or Host) to validate the Host ID and perform revocation through the CA System.

3.2.1 ID Report-back Mechanism

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* are sent to the Headend for validation using an authenticated message.

In a one-way system, POD will send the *Host_ID* and the *POD_ID* to the Host in the clear. The Host will respond with a confirmation message indicating that the message has been displayed to the customer. The Host will display the *Host_ID* and *POD_ID* in decimal format (digits 0-9) grouped in three digit sets so that they are easier to read and can be entered from a telephone dual-tone keypad. The end-user shall be required to telephone the service provider and report the *Host_ID* and *POD_ID* prior to receiving protected content. Alternatively, the retailer may perform this service for the end-user at time of purchase.

An example of the displayed message follows:

“In order to complete the pairing operation for this device, Please contact customer service at 1-800-xxx-xxxx with the following information:

*POD_ID=xxx-xxx-xxx
Host_ID=xxx-xxx-xxx”*

Customer service will interface with the headend for validation and authenticity of the *Host_ID* and *POD_ID* by verifying that services of the *Host_ID* have not been revoked. If the Host device is valid, the headend sends the *Host_ID* back to the POD via an authenticated message as *Validated_Host_ID* through the operator’s control stream.

Until the POD receives the validated *Host_ID* from the Headend, the POD shall not authorize any service with a EMI value of 01, 10, or 11, even if services so labeled are authorized in an EMM.

Additionally, the POD will validate the authenticity of the 5C-POD Device Certificate using the License Administrator’s public verification key that is delivered by the CA System to the POD in an authenticated manner. If the *Validated_Host_ID* value is the same as the *Host_ID* in the authenticated 5C-POD Device Certificate, the POD can continue with the key exchange process described in chapter 3.

3.2.2 Authentication Step 1 – Certificate Verification & DH Key Exchange

At the first step of the POD CPS authentication protocol, the Host certificate, *POD_ID*, 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 signature contained in the Host certificate can be verified by the POD. If the Host certificate verifies, the Host ID can then be extracted from the certificate. If any part of the copy protection system fails, including certificate verification, the POD shall not perform the CAS 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 HTML message to the Host similar to that shown below in Figure 1-C and shall notify the headend CAS (if possible).

There was a technical problem during the authorization process.

This product may have some component failure or may not be designed to be fully compatible with digital cable television services. Please contact the manufacturer or the retailer.

Figure 1-C, Copy Protection System Failure Notification Message

Thereafter, the invalid certificate notification screen (Figure 1-C) shall be displayed only if the verification of the Host 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 Step 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, a portion of the Diffie Hellman public keys, and a shared secret key (SSK). (A shared secret DH key is also derived in both POD and Host, using the DH public keys and the conventional DH protocol.)

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 5 seconds of the request message, the POD shall not perform the CAS 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 holds the same secret key as the POD. The exact same Authentication Key must be derived from the secret key (SSK).

3.2.4 Authentication Step 3 – Headend Report Back

Host authentication is performed via the Host ID validation check conducted by the headend. Depending on whether the protocol is executing in a one-way or two-way system, the POD Module or Host will request the Cable headend to validate the Host IDs. The Host ID validation process requires the CA System to check if the Host ID is listed in the CRLs stored in the headend.

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 Step 2 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.

3.2.4.1 One-way System Host ID Registration and Validation

After the POD Module verifies the Host certificate, the Host ID can be extracted from the Host's 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 communicates to the headend using private messages 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 certificate so it can be compared against the Host 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) so that these IDs can be entered from a telephone dual-tone keypad, if required.
- 4) The user must telephone the service provider and read the *Host_ID* and *POD_ID* from the screen to a custom service representative (CSR).
- 5) The CSR records the pairing between the *Host_ID* and *POD_ID*, as received from the user over the telephone. The CSR sends the *Host_ID* validation request to the CA System in the headend. The headend CAS records the pairing between *Host_ID* and *POD_ID*.
- 6) The CA System holds DTLA Certificate Revocation Lists and checks if the *Host_ID* is listed as revoked.
 - a) If the *Host_ID* is not found in the CRL, the CAS 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 CAS is allowed to authorize the POD for high value services with EMI values equal to 01, 10, or 11. See chapter 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 CAS shall mark that *Host_ID* as fraudulent, and is prohibited from authorizing that Host's associated 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 authorizing¹⁰ 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 Host_ID validation message and then compares the Host ID with the Host ID extracted from the certificate at step 1 above. The authentication is to verify that the message did originate in the headend.
- 9) The POD shall store the Host 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 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 Chapter 5.

The POD must 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.

3.2.4.2 Two-way System 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 (POD_ID) and the Host ID (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 still 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 certificate so it can be compared against the Host ID received back from the headend in step 3 below.
- 2) The POD sends the Host_ID and POD_ID through the upstream OOB in a private authenticated CA System message. The headend CAS records the pairing between Host_ID and POD_ID.
- 3) The CA System holds DTLA Certificate Revocation Lists (CRLs) and checks if Host_ID is listed as revoked.
 - a) If the Host_ID is not found in the CRL, the CAS 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 CAS is allowed to authorize the POD for high value services with EMI values equal to 01, 10, or 11. See chapter 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.

¹⁰ 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 CAS decryption step (even if the subscriber would otherwise be authorized to receive the service).

- c) If the Host_ID is found in the CRL, the CAS shall mark that Host_ID as fraudulent, and is prohibited from authorizing that Host's associated POD for high value services.
- 4) 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.
- 5) The POD authenticates the Host_ID validation message and then compares the Host ID with the Host ID extracted from the certificate at step 1 above. The authentication is to verify that the message did originate in the headend.
- 6) The POD shall store the Host 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 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 chapter.

3.3 Power-up Re-Authentication

At power-up, if the POD detects it holds a valid Authentication Key in non-volatile memory, the POD shall attempt a re-authentication procedure. This procedure will determine if the Host is the same 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 DH Keys and authentication keys are preserved, and only the Copy Protection Key is regenerated.

3.4 CP Key Refresh

3.4.1 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 CAS. 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 key exchange. 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**.

Table 3.4-A Key Session Period

Operation Parameter	# of bits	Comments
<i>max_key_session_period</i>	16	The time period for this parameter is in 10's of seconds. If the value of this parameters is '0', the maximum key session period is unlimited. The value for this parameter is established by the CA System and configured in the POD. The Host is not aware of this value.

3.4.2 Key Refresh Period

= In single CP_Key operation when a key refresh occurs, the POD module may start a one second timer and turn off scrambling of the MPEG content output and set the **CP_transport_scrambling_control_field** to 00 as defined in section 2.8.2.2. The Host receives cleartext packets and will not decrypt these packets according to MPEG rules. After completion of the key generation process the POD will immediately return to scrambling of MPEG packets. If the key exchange has not completed when the one second timer expires, the POD module shall disable CA decryption of copy protected content until a full CP_Key refresh is completed. The Host shall be implemented such that it supports the key exchange within one second. The Host shall also be implemented such that it can support key refresh every 2 seconds.

In dual CP_Key operation all protected content will 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.

3.4.3 CAS Key Refresh

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

3.4.4 Key Refresh Initialization

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

3.4.5 Channel Change

When a channel change occurs, the Host shall initially set EMI to 1,1, "copy never", and APS to 0,0, until the new CCI is received. The POD module shall determine and deliver the true value of the CCI to the Host. Channel changes do not require a key refresh to occur. Immediately after receipt the Host shall begin using the new CCI value.

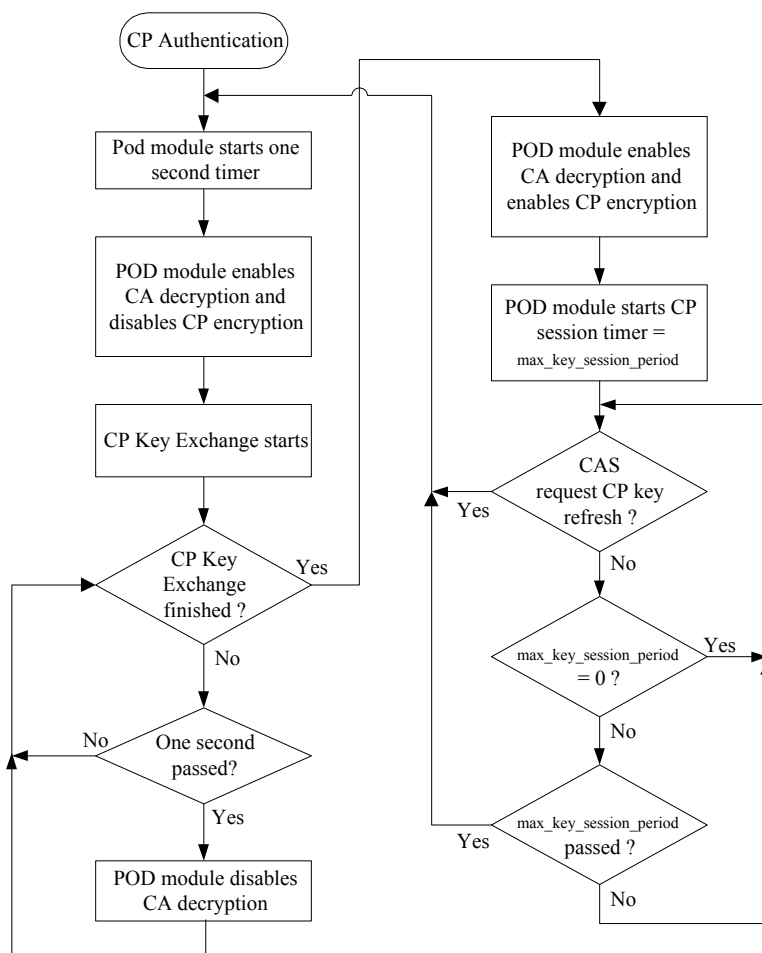


Figure 1-D CP Session Flow Chart

3.4.6 Two Key Synchronization Mode (Informative)

Optionally, a POD or Host may implement key refresh using a system of EVEN and ODD CP_Key's. When both POD and Host provide dual CP_Key support it shall be used and the one second clear period shall not be used. 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 so implemented the transport_scrambling_control bits shall indicate use of the EVEN or ODD CP_Key registers as shown in table 14.. It is highly recommended that POD and Host vendors 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.

3.5 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 Copy Protection 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 will initiate the binding procedure, including full 5C-POD certificate authentication. If this POD is later returned to the previous Host, and it will again initiate the binding procedure as it has authentication information only on the last Host to which it was bound.

3.6 Host Operation with Multiple PODs

Each Host shall bind to exactly one POD at a time. No Host shall store two or more sets of Copy Protection Keys or other POD-specific information. A Host shall participate in re-initialization of the binding procedure upon request from a POD.

4. Cryptographic Functions

The basic key negotiation process for POD copy protection is shown in Figure 2 below.

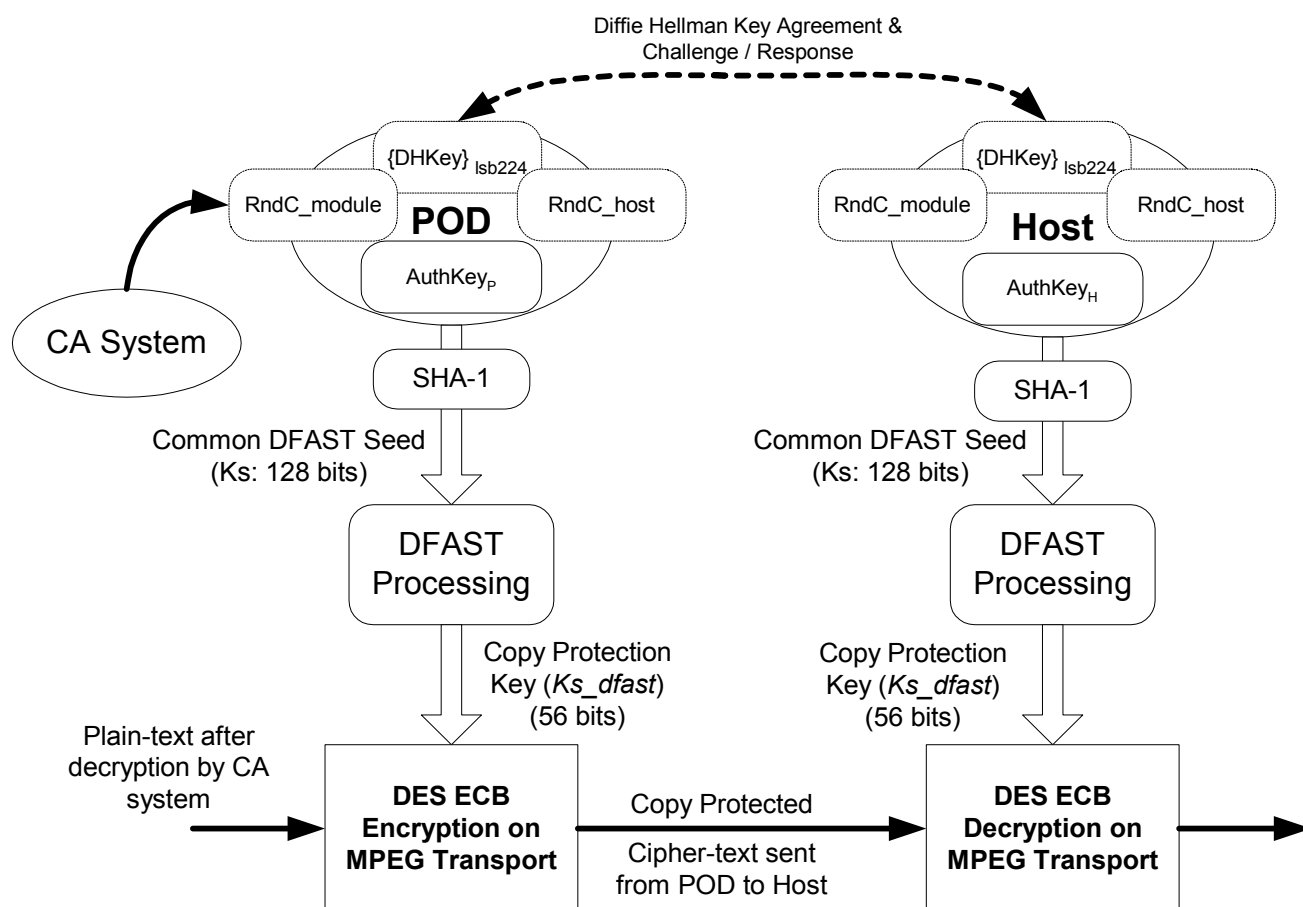


Figure 2 Overall POD Copy Protection

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 I_{ds} 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 140 bits of each DH parameter, the 64 bit POD Module ID, the 40 bit

Host ID, and a 128 bit shared secret called SSK. 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 POD computes its authentication key by applying the SHA-1 function:

$$AuthKey_P = \text{SHA-1} [SSK_P \mid DH_pubKey_P_{1sb140} \mid DH_pubKey_H_{1sb140} \mid Host_ID \mid POD_ID]$$

The 140 least significant bits of the DH public keys are used ($= \{512 - 128 [SSK_H] - 40[Host_ID] - 64 [POD_ID]\} / 2$). SSK_P is the Cablelabs-licensed shared secret key stored in the POD to authenticate the DH protocol. SSK_H is the Cablelabs-licensed shared secret key stored in the Host to authenticate the DH protocol. SSK_P and SSK_H have the same 128 bit numeric value, and are obtained from Cablelabs along with DFAST information.

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

$$AuthKey_H = \text{SHA-1} [SSK_H \mid DH_pubKey_P_{1sb140} \mid DH_pubKey_H_{1sb140} \mid Host_ID \mid POD_ID]$$

$AuthKey_P$ and $AuthKey_H$ are used in Copy Protection Key Generation, as described in section 3.2.2.

Authentication Key generation need occur only once (per POD/Host pair) when the POD and Host is first connected. The resulting $AuthKey_P$ and $AuthKey_H$ values 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

The Copy Protection Key generation process is conducted to derive a new Copy Protection Key (i.e. a new Ks_dfast). This occurs at the following times:

- At the end of the authentication process;
- 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 packets sent from the POD to the Host.

4.2.1 Basic Key Generation Protocol

To generate the content encryption and decryption key in POD CPS, a simple key generation procedure is used. In addition to the long-term Authentication Key and 1024 bit shared secret Diffie-Hellman key ($DHKey$) derived from the Host authentication process, new random numbers are generated and included as a nonce in the key generation, see details in the following subsections.

To generate a new Copy Protection Key, the following procedure shall be used:

- 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 starts a one-second time-out period and turns off encryption of the MPEG content.
- 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 ID (*Host_ID*) in the clear to the POD.
- 7) The POD checks if the received *Host_ID* is equal to the previously stored ID. If they are the same, POD shall proceed with the key generation process; otherwise, the POD shall only authorize services with EMI values of 00.
- 8) 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.
- 9) 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*:

$$Ks = \text{SHA-1} [AuthKey_P \mid \{DHKey\}_{1sb224} \mid N_{Host} \mid N_{module}]_{1sb128}$$

The field $\{DHKey\}_{1sb224}$ has length 224 bits (= 512 – 160 (*AuthKey_P*) – 64 (*N_{Host}*) – 64 (*N_{module}*)). Detailed information on how to generate *AuthKey_P* is described in section 4.1. SHA-1 is used as a cryptographic compression function to generate a seed with the proper 128 bit length for the input to the DFAST¹¹ engine. The DFAST algorithm is applied to *Ks* to produce the 56-bit value of the Copy Protection Key, also known as *Ks_{dfast}*:

$$\text{Copy Protection Key} = Ks_{dfast} = \text{DFAST} [Ks]$$

DFAST details are specified in a separate document; contact CableLabs. Table 17 defines the size of keys, as well as the parameters used to derive them.

¹¹ CableLabs has made arrangements to license the DFAST technology (U.S. Patent number 4,860,353 and related know-how) necessary to implement the OpenCable POD Module Interface Technology, to any interested party under fair and reasonable terms. For licensing information, refer to the OpenCable website <www.opencable.com> or contact CableLabs at 303/661-9100. OpenCable POD Module Interface Technology is defined as all relevant POD Module interface specifications in addition to the DFAST intellectual property.

Table 17 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 Authentication 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 ($DHKey$) _{lsb224}	224 bits	The least 224 significant bits of the 1024 bit shared DH secret key. It is a long-term key, and is stored in non-volatile memory.
SHA-1 Key K_s	128 bits	The least 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 (K_s_{dfast})	56 bits	DFAST output, final content encryption and decryption key

4.2.3 Host Copy Protection Key

The Host computes its SHA-1 key based on the Authentication Key ($AuthKey_H$), the 224 lsb's of the 1024 bit shared secret DH key ($DHKey$), and the random numbers exchanged in the key generation. This key is named K_s :

$$K_s = \text{SHA-1} [AuthKey_H \mid \{DHKey\}_{lsb224} \mid N_{Host} \mid N_{module}]_{lsb128}$$

Detailed information on how to generate $AuthKey_H$ is described in section 4.1. SHA-1 is used as a cryptographic compression function to generate a seed with the proper length for the DFAST engine. The DFAST algorithm is applied to K_s to produce the 56-bit value of the Copy Protection Key:

4.3 Copy Protection Key Refresh

The CP key will 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, and starts a one second timer, it may stop scrambling the selected program. It will start to encrypt again on the earlier of; successful completion of the authenticated CP key refresh cycle, or timeout of the one second timer. The CCI will not be changed during this <1 second period.

Each CP Key refresh will 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 $AuthKey_H$ to its stored $AuthKey_P$ 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.)

4.4 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 content encryption/decryption key generation. The Diffie-Hellman protocol provides the system with a cryptographic property known as “perfect forward secrecy”. Figure 3 illustrates the three-step Diffie-Hellman operations conducted in the POD, Host and interface between them.

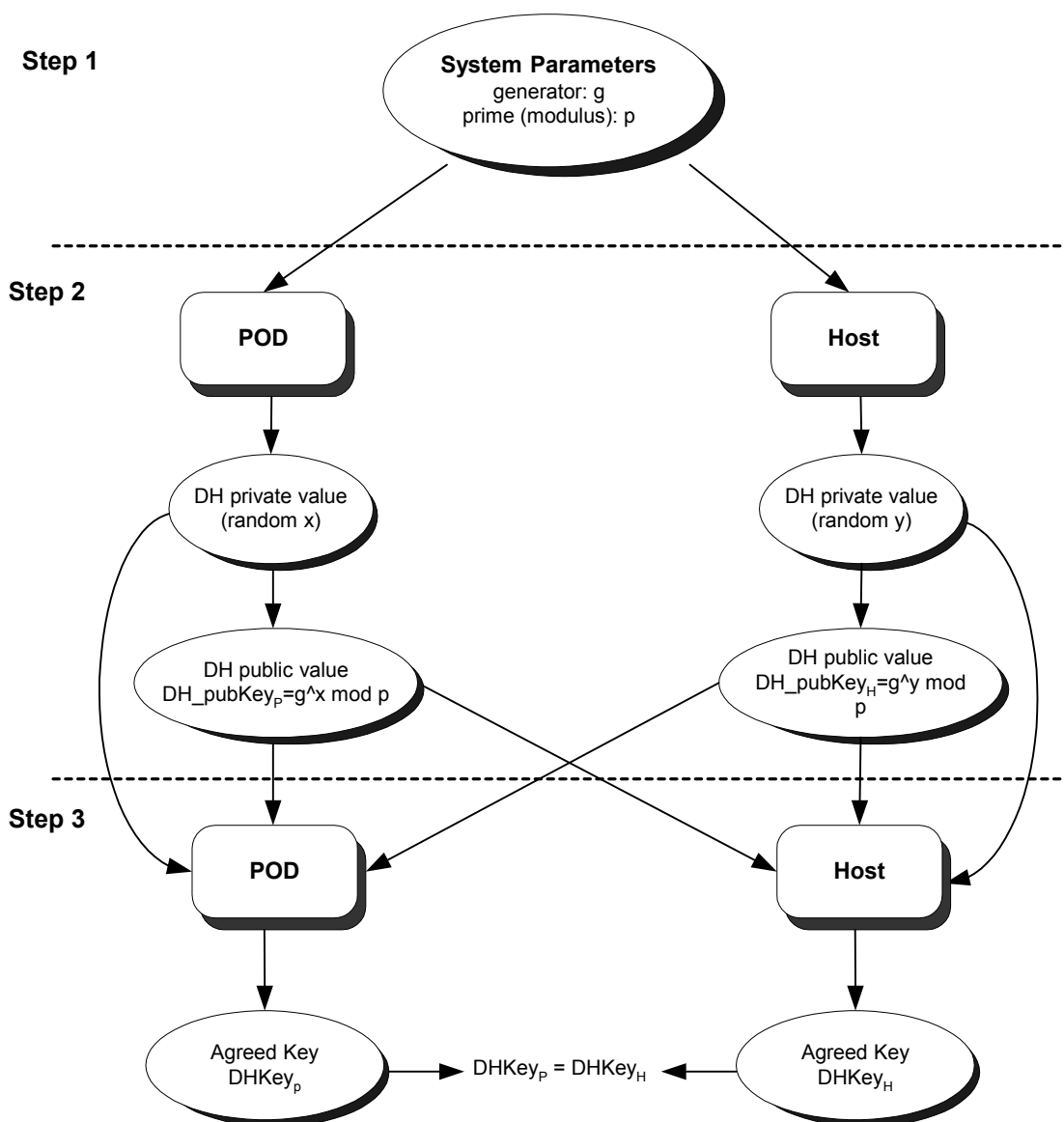


Figure 3 Diffie-Hellman Key Agreement Protocol between POD and Host

4.4.2 Algorithm Implementation

4.4.2.1 System Parameter Generation [Informative]

The length of the modulus (p) 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 to the manufacturer by CableLabs when the license to implement OpenCable POD Module Interface Technology is granted.

4.4.2.2 Step 2 Operations

Each of the two parties (POD and Host) executing the Diffie-Hellman protocol does the following:

1. The POD randomly generates a private exponent, x , where $0 < x < p$, 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 $0 < y < p$, selecting y to have the length of at least 160 bits.
3. The POD computes its public key value $DH_pubkey_P = g^x \text{ mod } p$, and sends it to the Host along with other authentication data.
4. The Host computes its public key value $DH_pubkey_H = g^y \text{ mod } p$, and sends it to POD along with other authentication data.
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 3 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 p , as follows:

- 1) The POD derives the 1024 bit shared key $DHKey_P = (DH_pubkey_H)^x \text{ mod } p$; and
- 2) The Host derives the 1024 bit shared key $DHKey_H = (DH_pubkey_P)^y \text{ mod } p$;
- 3) Even though both the POD and Host are making computations using different private values (x, y), they end up with the same secret key: $DHKey_P = DHKey_H$ because:

$$DHKey_P = (g^y)^x \text{ mod } p = g^{(yx)} \text{ mod } p = (g^x)^y \text{ mod } p = g^{(xy)} \text{ mod } p = DHKey_H$$

4.5 SHA-1 Secure Hash Algorithm

SHA-1, as described in FIPS PUB 180-1 is the algorithm used in the Digital Signature Algorithm (DSA) to generate a message digest of length 160 bits.

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

- Host Certificate Signature Verification: the signature algorithm is based on the ECC-DSA digital signature scheme defined in FISP 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 *RndC_module*, *RndR_Host*, *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. Each OpenCable device shall 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 Elliptic Curve Digital Signature Algorithm (EC-DSA) [Informative]

The Elliptic Curve Digital Signature Algorithm (ECDSA) is the elliptic curve analogue of the DSA. It is used to provide data authentication, data integrity, and non-repudiation. ECDSA is approved as ANSI standard – ANSI X9.62 and IEEE P1363.

4.7.1 Elliptic Curve Domain Parameters [Informative]

Elliptic Curve domain parameters contain:

- Prime number p of finite field $GF(p)$
- Coefficients of EC polynomial (a, b)
- Base point G of prime order n
- Cofactor: $h = \#E(F_p) / n$ where $\#E(F_p)$ is the number of points on the elliptic curve.

Contact DTLA for these system parameters.

4.7.2 EC Key Pair Generation [Informative]

Each entity A does the following:

- Select a random integer d in the interval $[1, n-1]$
- Compute the public key $Q = dG$;
- A 's public key is Q ;
- A 's private key is d .

DTLA shall provide its public key to all the POD manufacturers.

4.7.3 ECDSA Signature Generation [Informative]

To sign a message, A does the following:

1. Select a random integer k , $1 \leq k \leq n-1$.
2. Compute $kG = (x, y)$, and $r = x \bmod n$. If $r = 0$ then go to step 1.
3. Compute $k^{-1} \bmod n$.

4. Compute $e = \text{SHA-1}(m)$.
5. Compute $s = k^{-1} \{e + dr\} \bmod n$. If $s = 0$ then go to step 1.
6. A 's signature for the message m is (r, s) .

4.7.4 ECDSA Signature Verification [Normative]

To verify A 's signature (r, s) on m , B should do the following:

1. Verify that r and s are integers in the interval $[1, n-1]$.
2. Compute $e = \text{SHA-1}(m)$.
3. Compute $w = s^{-1} \bmod n$.
4. Compute $u_1 = ew \bmod n$ and $u_2 = rw \bmod n$.
5. Compute $u_1G + u_2Q = (x, y)$ and $v = x \bmod n$.
6. Accept the signature if and only if $v = r$.

4.8 DFAST Algorithm

4.8.1 Algorithm Overview

The diagram in Figure 3 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 CableLabs.

4.8.2 DFAST Characteristics

The functionality of DFAST can be summarized as follows:

- DFAST is a U.S. and internationally patented algorithm;
- DFAST is an efficient, practical and easily implemented keystream generation;
- DFAST can produce sequences with a large period, and with good statistical properties;
- DFAST is a highly non-linear keystream generation algorithm; and
- DFAST takes 128 bits of input (Ks) and generates 56 bits of output (Ks_{dfast} , or the Copy Protection Key). The output from DFAST is used as the DES ECB key for copy protection content encryption and decryption.

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

Identifying the circumstances where revocation must be possible helps us determine where revocation should occur and how to introduce and specify the revocation mechanisms in the system. The revocation technique must be used as a safeguard to prevent copy protected services from being passed to a compromised Host. Cases where this might occur include:

1. A fraudulent Host claimed to be compliant with CableLabs specifications and standards;
2. An erroneously-designed Host claimed to be compliant with CableLabs specifications and standards;
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 debugging programs or a specific security scheme that has been compromised).
4. A Host clone or pirate clone.

5.3 Fraudulent Host Identification

This POD-CP system requires the POD Module to retrieve the Host certificate from the Host. The POD then reports this certificate and the POD_ID to the headend when a POD Module is mated 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 or voice).

In all cases the Host ID shall be reported back to the headend by some means. Except for the case of a copy protection system failure or an invalid Host Certificate (*Cert_H*) 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 performed locally in the POD or Host is unnecessary, and is therefore not reflected in this document.

5.4 CAS Revocation & Selective Denial of Services

5.4.1 Definition of Revocation

Revocation is the concept of limiting or zeroing the usefulness of a suspect Host in the network. This is achieved by controlling the value of services that Host is authorized to receive.

The easiest mechanism to achieve this is the one that has already been performing this function for years, the Conditional Access System. It is obvious that each Host receiving high value services necessarily has a POD immediately upstream of that device, or said Host would not operate usefully. If that upstream POD does not decrypt the services sent it from the CA System Headend, then the associated Host cannot operate. Therefore, revocation of any selected service for that Host can be achieved merely by knowing the identity of its associated POD, and need not be any more complex.

Knowing the pairing of the POD to the Host is central to the use of CA System revocation. In a two way system this is done using the upstream capability of the POD, which reports the Host ID of the Host it is inserted into during POD-Host initialization. In a one way system, this is achieved by the consumer reporting the Host ID to the CA System Headend CSR, and by that same Host ID being echoed back to the POD in a secure manner. Thus, in both one way and two way systems, the CA System headend can reliably determine the Host ID associated with its PODs, and can therefore revoke selected services of Hosts based on this knowledge.

5.4.2 Selective Service Denial

Fundamentally, services are revoked, not a physical Host. A fraudulent Host can still legally watch clear services, for example, as well as free special offerings. *Revocation really means a scheme 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 an 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 17 in chapter 6. 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”). A EMI value of 00 (“Copying is Permitted”) is not considered to be high value content.

The status or trustworthiness of a Host can be in question due to a number of events:

- Event 1. The Host may not have completed its authentication protocol, due to the POD failing to verify the Host certificate.
- Event 2. The Host may not have completed its authentication protocol, due to the Authentication Key not being confirmed in the POD.
- Event 3. The Headend may not have confirmed that the Host is 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.
- Event 4. The POD finds itself connected to a new, unregistered and unauthenticated Host at power-up, and is unable to complete the entire authentication protocol, similar to 1 above.

If events #1, 2 or 4 occur, the POD shall not perform the CAS decryption step, shall send the appropriate MMI message to the Host and shall notify the headend in a two-way plant as defined in Section 3.2.2 above. If event #3 occurs, the POD shall not authorize for high value services with EMI equal to 01, 10, or 11. The POD may authorize for EMI equal to 00, if so commanded by the CAS.

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 MPAA, DTLA, and the MSOs. See the DTLA agreement for details.

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 using CA System EMMs, so no CRLs need exist in the cable network. Further, the CA System headend can receive new CRLs, look up new Host IDs, and then deauthorize specific services granted to the POD associated with any bad Hosts at any time.

Table 18 CRL Based Host Service Revocation

One-way System (Voice)	Two-way System	Description
ECC-DSA based Certificate Verification Voice ID report-back mechanism ID vs. CRLs check and return-back by the headend at any time Revocation display message to user	ECC-DSA based Certificate Verification ID report-back mechanism ID return-back by the headend at any time Revocation display message to user	Compromised Host is listed in CRLs provided by DTLA, MPAA, or service providers. CRL based revocation is built in the CA System in the headend. If ID is 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 such as a 1394 port.

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 will ignore the reserved bits values thereafter.

Table 19 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	x	x	x	x	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 including the 1394 source port for control of copies made from those outputs. The EMI bits are defined in Table 20.

Table 20 EMI Values and Content

EMI Value	Digital Copy Permission	Value of Content with this Label
00	Copying is permitted.	Not High Value
01	No further copying is permitted.	Possibly High Value
10	One generation copy is permitted.	Possibly High Value
11	Copying is prohibited.	Definitely High Value

6.1.2 APS - Analog Protection System

The Host shall use the APS bits to control copy protection encoding of analog composite outputs as described in Table 12 of EIA-679B part B.

6.2 Associating CCI with a Service

The CAS 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 CAS 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 7. The messages are based on a SHA function performed on the CCI, CP_Key and MPEG program number.

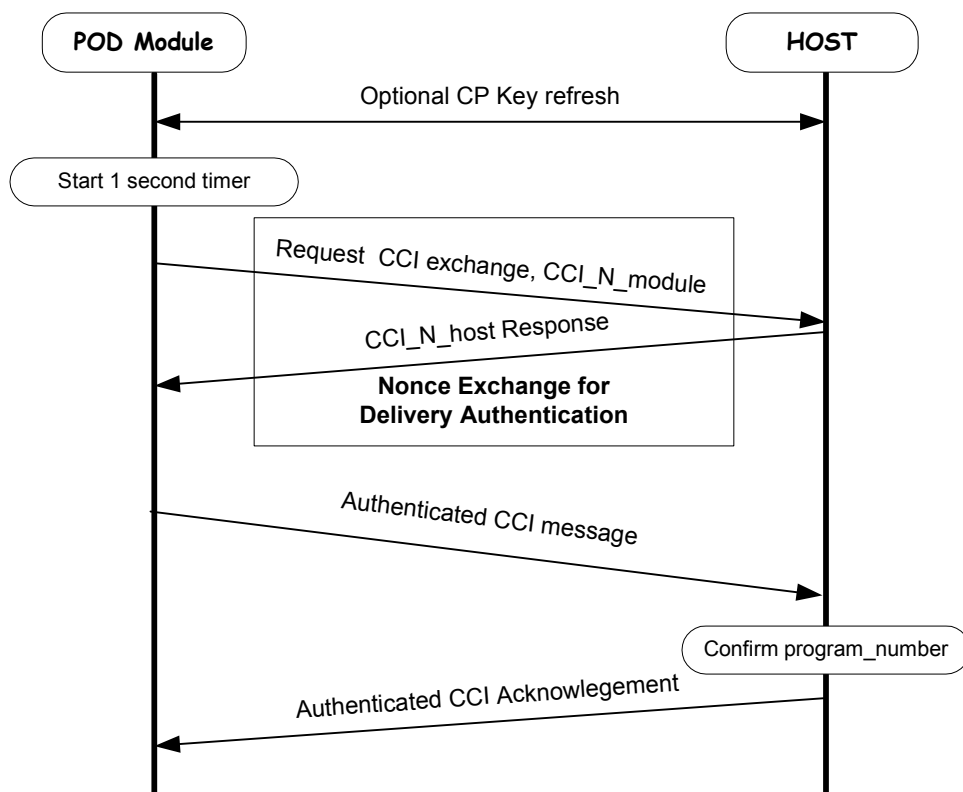


Figure 7 CCI Delivery Sequence

6.4.1 CCI Delivery Instances

CCI is sent from POD to Host only after the POD and Host have successfully mated and negotiated a shared CP_Key. CCI transfer from the POD to the Host shall initiate immediately after any of:

1. the POD is tuned to new 'channel' by the Host, or

2. the MPEG program number changes on a tuned 'channel', or
3. the CCI value changes during a program, or
4. the MPEG packet ID values that the POD is decrypting change.

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. Until a successful CCI delivery has completed the POD shall disable CA-descrambling of protected content and the Host shall default to maximal protection of all CP-scrambled content.

- Step 1. The POD generates a new random number CCI_N_module and starts a one 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 the CCI_N_host and program_number (as 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:

$$\text{CCI_auth} = \text{SHA-1}(\text{CCI} \mid \text{CP_Key} \mid \text{CCI_N_module} \mid \text{CCI_N_host} \mid \text{program_number})$$

$$\text{CCI_ack} = \text{SHA-1}(\text{CCI} \mid \text{CP_Key} \mid \text{CCI_N_module} \mid \text{CCI_N_host})$$
- 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.
- 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.

NOTE: If the steps above are not completed within one second during single key operation, the POD will timeout and disable delivery of copy protected content on the POD-Host interface until the CCI delivery protocol completes successfully.

7. Transport Encryption From POD to Host

MPEG content which is delivered with free rights to copy, e.g., free access off-air broadcast content, will be delivered in the clear on the POD Host Interface. Such content may or may not be encrypted on the delivery from the headend to the POD. Such content is marked CCI = 0 for all outputs and is not protected by the means described in this specification.

7.1 MPEG DES Encryption

DES ECB shall be used to encrypt copy protected MPEG services in the POD and to decrypt 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 to "scrambled".

POD CP encryption shall only be applied to those MPEG PIDs which were decrypted by the POD CA decryption function. Clear services, CA encrypted but unauthorized services, and CA encrypted and authorized but unselected services shall pass through POD unaltered.

POD CP encryption shall only be applied to packets output from POD when those packets are part of an authorized service. If a service is suddenly deauthorized, then POD CP encryption (along with CA decryption) shall be deactivated as described in section 7.3 below. (A consequence of this is that no packet shall be double encrypted, once using CA and again using CP. Such double encryption would be an implicit MPEG violation.)

7.2 Transport Processing

For a given MPEG packet output from POD encrypted using copy protection DES ECB, the packet encryption parity may take on the values 0 or 1 without limitation. The value of the MPEG packet scrambling bits shall be limited to 11, denoting an encrypted packet. When the POD outputs clear, unencrypted packets, the parity and scrambling bits shall both be set to zero.

CP encryption mode changes (i.e. transitions from "CP Encryption ON" to "CP Encryption OFF/Clear") shall be handled so as to avoid any period of time where clear MPEG packets are sent from POD to Host, except where said packets were already clear when input to the POD, or for a minimal time during single CP_Key refresh. If the status of CA decryption changes (e.g. due to a change in POD entitlement or a change in the encryption mode of the MPEG stream input to the POD), then the POD shall alter the mode of its input CA decryption prior to altering its mode of output CP encryption.

7.3 Timing of Encryption Transitions

CP encryption mode changes from "CP Encryption OFF" to "CP Encryption ON" shall be accomplished quickly and in no case more than 1.5 seconds after the event that causes the mode change, e.g., a EMI change from 0 to 1,2,or 3. All MPEG packets of the relevant program shall be output from the POD CP-encrypted as soon as possible following a EMI change from 0 to a protected value of 1, 2, or 3, likewise a change to EMI=0 shall be accomplished within 1.5 seconds.

CA decryption mode may change due to a change in POD entitlement or a change in the encryption mode of the MPEG stream input to the POD. The POD will continue to encrypt all MPEG packets of a CP protected program during any such changes in CA operation unless either CA encryption mode changes to clear or EMI changes to 0.

7.4 Debug Modes Prohibited

Production POD's or Hosts shall have no debug modes that bypass or compromise security provisions of this interface.

7.5 Copy Protection Encryption as a function of CA Encryption and EMI Value

The POD shall apply copy protection encryption of content flowing to the Host as shown in Table 21.

Table 21 CP Encryption of Content based on CA Encryption State and EMI Value

CA Encryption State	EMI Value	POD Encrypts Output	Comments
Clear	0	No	
Clear	1, 2, or 3	No	Undesired*
Encrypted	0	No	
Encrypted	1, 2, or 3	Yes	

* Cable operators should CA encrypt all programs with EMI>0. Only programs with CA encryption active will be protected from unauthorized copying.

8. Host, POD, & Headend Messaging Protocols

8.1 Message Protocol Overview

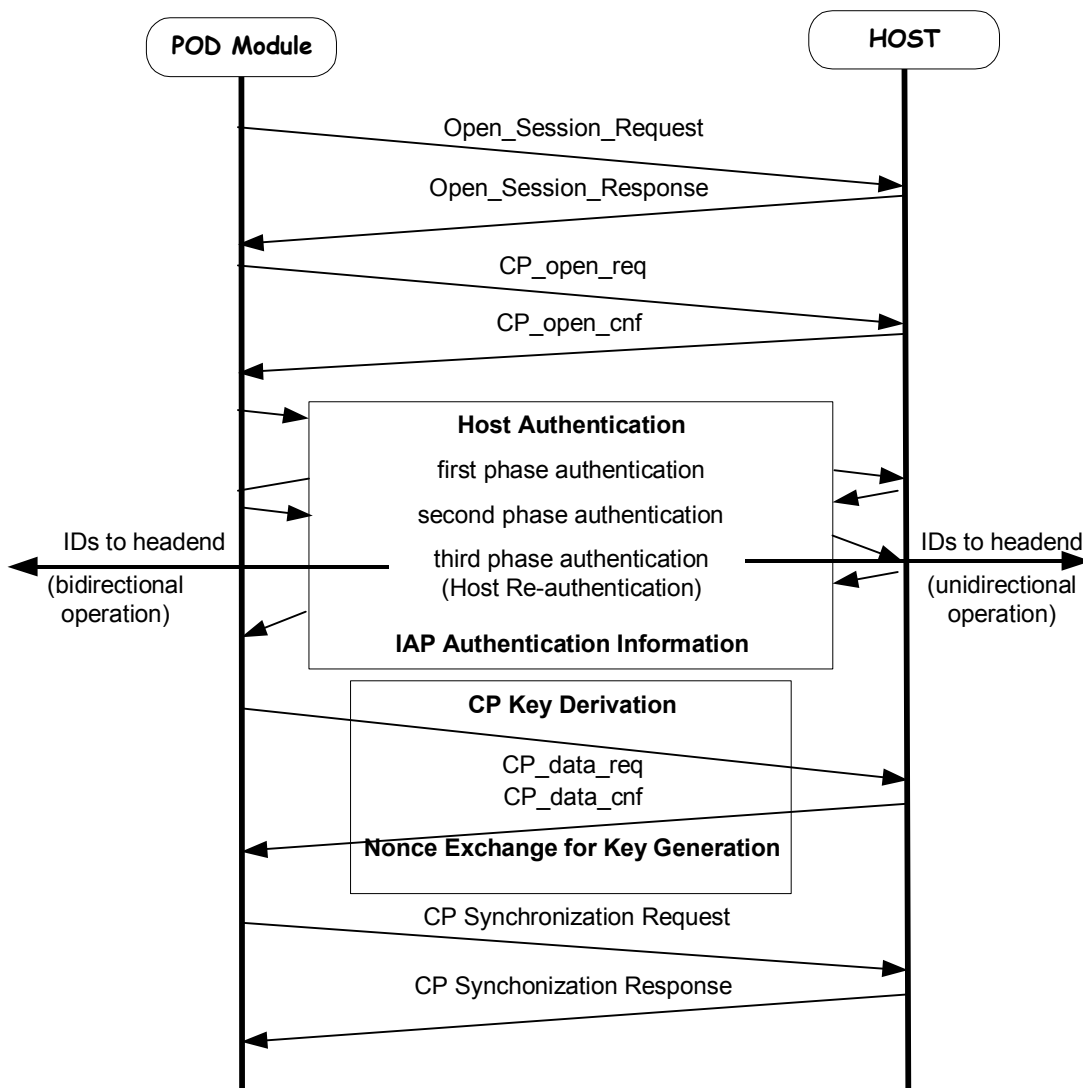


Figure 5 Copy Protection Message Protocol Overview

Figure 5 gives an overview of content copy protection protocol. In this overview, messages start from the POD after the Host recognizes it.

- 1) An application in 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 or to tell the module that its request could not be fulfilled.
- 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 system (POD CPS) it supports.
- 3) The POD, if it holds a valid Authentication Key in non-volatile memory, will attempt to perform a re-authentication procedure by requesting the Host's Authentication Key (third phase authentication.) If the POD's and the Host's Authentication Keys match, 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 encrypt the content at the POD Module and decrypted the content at 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 EIA-679-A (Part B)

Table 22 Copy Protection Open Session Information

SPDU Tag / Object	Tag Value (Hex)	Action	Direction
<code>Open_Session_Request()</code>	91	The POD Module requests a session of the Copy Protection resource to be opened.	POD → Host
<code>Open_Session_Response()</code>	92	The Host responses with a session status, if successful. A session number is assigned. The session number will 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 23 Open_Session_Request() Message Syntax

Message Syntax	bits	bytes	Description
<pre> open_session_request () { Open_session_request_tag Length_field() Resource_identifier() } </pre>	8	1	Has the value of 91 (hex)
	8	1	length_field () is defined in EIA-679-A (Part B) section 7. Since resource_identifier() followed only has 32 bits, length_field() shall have the following values set: size_indicator = 0, length_value = 4
	32	4	Resource_identifier () is defined in EIA-679-A (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 section 7.2.6.1 of EIA-679-A, Part B: 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 4.

Table 24 (removed – see Table 4)

If the version field of the supplied resource identifier is zero, then the Host will 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, then the Host will refuse the request with the appropriate return code, as defined in EIA-679-B (Part B).

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 25 Open_Session_Response() Message Syntax

Message Syntax	bits	bytes	Description
<pre> open_session_response () { Open_session_response_tag Length_field() Session_status Resource_identifier() Session_nb } </pre>	8	1	Has the value of 92 (hex)
	8	1	length_field () is defined in EIA-679-A (Part B) section 7. Since session_status (1 byte), resource_identifier() (4 bytes), and session_nb (2 bytes) are followed, length_field() shall have the following values set: size_indicator = 0, length_value = 7
	8	1	Session status values are listed in EIA-679B part B.
	32	4	Resource_identifier () is defined Table 4.
	16	2	The Host allocates session number for the requested session. Value 0 is reserved. The session_nb will 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.

Table 26 (removed)

8.2.2 Host Capability Evaluation

The NRSS Copy Protection Framework requires the POD Module to check the Host's ability to support the OpenCable™ Content Protection algorithm, 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 27 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 28 CP_open_req() Message Syntax

Message Syntax	bits	bytes	Description
CP_open_req () { CP_open_req_tag Length_field() }	24	3	Has the value of 9F9000 (hex)
	8	1	length_field () is defined in EIA-679-A (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 29 CP_open_cnf() Message Syntax

Message Syntax	bits	bytes	Description
CP_open_cnf () { CP_open_cnf_tag Length_field() CP_system_id_bitmask }	24	3	Has the value of 9F9001 (hex)
	8	1	length_field () is defined in EIA-679-A (Part B) section 7. The length_field() shall have the following values set: size_indicator = 0, length_value = 4
	32	4	Values are list in Table 30.

Where CP_system_id_bitmask value is defined in Table 30.

Table 30 CP_system_id_bitmask Values

CP_system_id_bitmask	Bit Number	Description
System 1	0	reserved
System 2	1	POD CPS
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

After full authentication or re-authentication has been performed a power-up, or at the end of a CP key session period, the POD module sends a request to the Host to generate a Copy Protection Key. The request contains the POD ID as well as a random nonce for use in generating the new key. Upon receipt of this request, the Host will also generate a random nonce and send it to the POD along with its Host ID. Both the POD and the Host will then generate a new CP key, using both random nonces and the 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 31 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 ($Host_ID$) 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 32 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 EIA-679-A (Part B) section 7. Since there is no other field followed, length_field() shall have the following values set: size_indicator = 0, length_value = 27
CP_system_id	8	1	Values are listed in Table 34: 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++) {	(32)	(2*3)	
Datatype_ID	8	1	When i = 0, Datatype_ID has the value of 6 (POD_ID)
	8	1	When i = 1, Datatype_ID has the value of 12 (N_module)
Datatype_length	16	2	When i = 0, Datatype_length has a value of 8
	16	2	When i = 1, Datatype_length has a value of 8
For (j=0; j<Datatype_length; j++) {	(128)	(16)	
Data_type	64	8	When i = 0, Data_type = <i>POD_ID</i> (see Table 33); and
	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 has the value of 5 (Host_ID)
	8	1	When i = 1, Datatype_ID has the value of 11 (N_Host)
}			
}			

The data type values (Datatype_ID) and sizes (Datatype_length) used in the above table is defined in Table 33.

Table 33 Datatype_ID and Datatype_length Values

Datatype_id	id value	Size (Bytes)
Manufacturer_id	1	50 (Maximum)
Brand_id	2	50 (Maximum)
Model_id	3	20 (Maximum)
Serial_id	4	20 (Maximum)
Host_ID	5	5
POD_ID	6	8
RndR_Host (random response generated by Host)	7	8
RndC_module (random challenge generated by the	8	8
New_Manufacturer_id	9	50 (Maximum)
New_Host_id	10	8
N_Host (Host's challenge to POD)	11	8
N_module (POD's challenge to Host)	12	8
DH_pubKey_Host (Host DH Public Key)	13	128
DH_pubKey_module (Module DH Public Key)	14	128
Cert_Host (Host Certificate Data)	15	88
Cert_module (Module Certificate Data)	16	Not Used in POD CP
Reserved	17	
Reserved	18	
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

CP_system_id used in the above table is defined in Table 34.

Table 34 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

8.2.3.2 CP_data_cnf() Syntax in Key Generation

This object also contains Host's ID and nonce so the POD Module can derive its CP content encryption key.

Table 35 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 9F9005 (hex)
Length_field()	8	1	length_field() is defined in EIA-679-A (Part B) section 7. The length_field() shall have the following values set: size_indicator = 0, length_value = 20
CP_system_id	8	1	Values are listed in Table 34.
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 has the value of 5 (Host_ID); and
	8	1	When i = 1, Datatype_ID has the value of 11 (N_Host)
Datatype_length	16	2	When i = 0, Datatype_length has the value of 6; and
	16	2	When i = 1, Datatype_length has the value of 8.
For (j=0; j<Datatype_length; j++)	(104)	(13)	
{			
Data_type	40	5	When i = 0, Data_type = Host_ID (see Table 33)
	64	8	When i = 1, Data_type = N_Host; and
}			
}			
}			

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 36 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 37 CP_sync_req() Message Syntax

Message Syntax	Bits	bytes	Description
CP_sync_req () { CP_sync_req_tag Length_field() }	24	3	Has the value of 9F9004 (hex)
	8	1	length_field () is defined in EIA-679-A (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 38 CP_sync_cnf() Message Syntax

Message Syntax	bits	bytes	Description
CP_sync_cnf () { CP_sync_cnf_tag Length_field() Status_field }	24	3	Has the value of 9F9004 (hex)
	8	1	length_field () is defined in EIA-679-A (Part B) section 7. The length_field() shall have the following values set: size_indicator = 0, length_value = 1
	8	1	

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 39.

Table 39 Status_field Value

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

Table 40 (removed)

Table 41 (removed)

Table 42 (removed)

Table 43 (removed)

8.3 One-way System IAP Message Protocol

8.3.1 Protocol Flow Overview

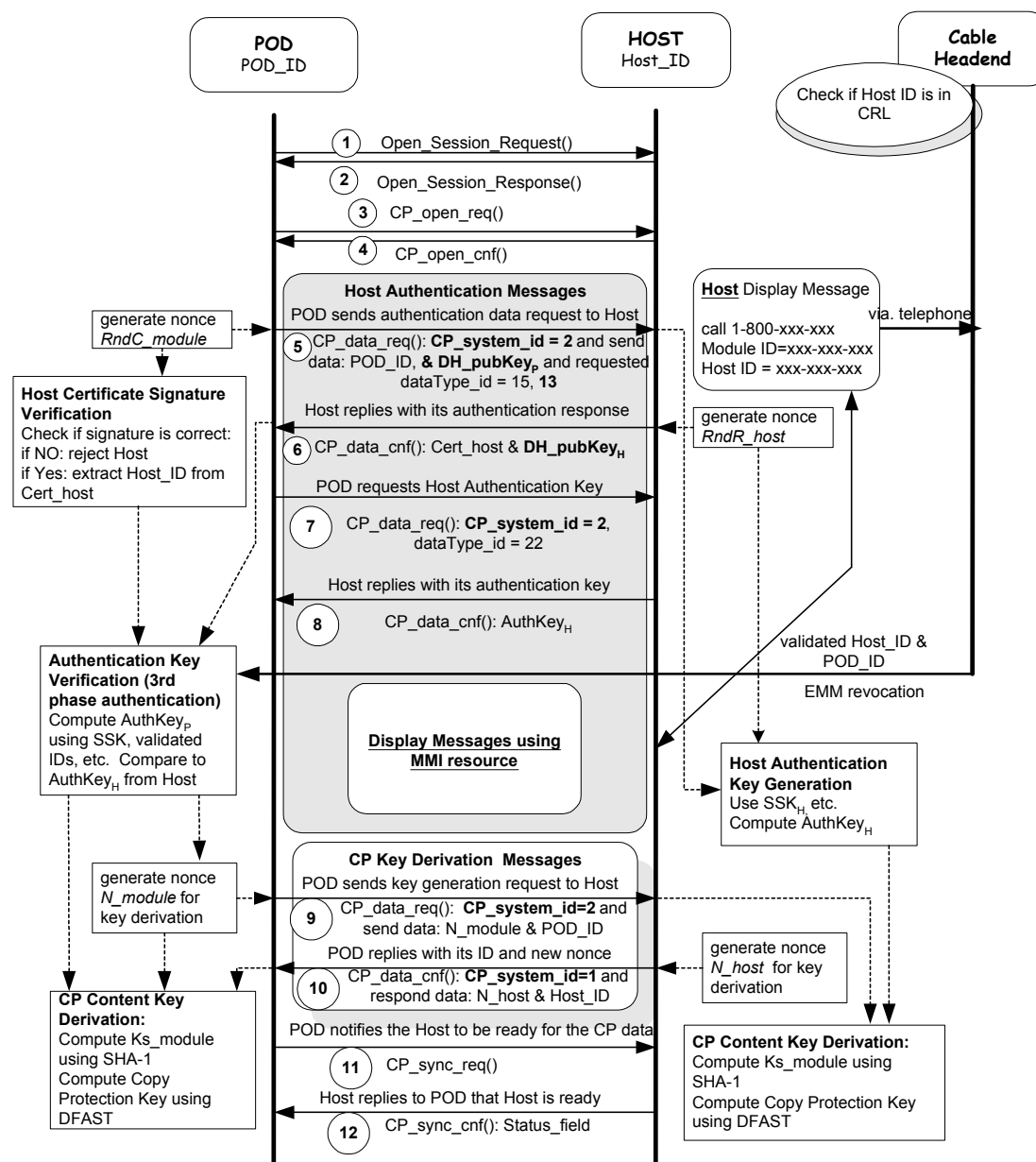


Figure 6 One way System POD CPS Message Protocol Overview

Figure 6 gives an overview of the POD CPS Protocol message flow in the one-way system. 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 44 One-way System POD CPSP 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 8.3.2.1below	Host authentication data
6	CP_data_cnf	APDU / 9F9003	Section 8.3.2.2 below	
7	CP_data_req	APDU / 9F9002	Section 8.3.4.1	Authentication Key verification
8	CP_data_cnf	APDU / 9F9003	Section 8.3.4.2	
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 / 9F9002	Section 8.2.4.1	POD & Host Synchronization
12	CP_sync_cnf	APDU / 9F9003	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 signature. The second step of authentication is based on headend being able to verify that the Host ID is not included in CRLs, and the POD being able to confirm the ID received from headend is the same as the one stored locally. The third 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.

At power-up, if the POD holds a valid Authentication Key, it will utilize the third step of authentication described above to attempt to perform a re-authentication (steps 12 & 13 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 and the POD can proceed with CP key generation. If the keys do not match, then a full Host authentication will be performed, starting with step 1 below. One-Way System POD CPS Protocol Steps Full Authentication

- 1) The POD Module initiates the authentication protocol by sending a challenge request to Host. This challenge request contains its ID (POD_ID) 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 sends its reply with its certificate (Cert_Host) and 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 Module checks if certificate is valid by:
 - Checking the value of the certificate type or format field; and
 - Verifying the DSA signature.
- 4) If Cert_Host is valid, then POD extracts Host_ID from the certificate.
- 5) The POD Module opens an MMI dialog to present the POD_ID and Host_ID to the subscriber, typically with a telephone number, for manual communication to the headend.
- 6) **(Host → Cable Headend)** The end-user will call the service provider and report the Host_ID and POD_ID via telephone; see section 3.2.4.1. Detailed operational requirements and message syntax are outside the scope of this document.
- 7) **(Cable Headend CRL Checking; second step of authentication)** Check if Host_ID is in the CRLs. This check may not occur in real time; see section 3.2.4.1.
- 8) **(Cable Headend - > POD)** Send EMM to authorize the POD; see section 3.2.4.1.
- 9) **(Cable Headend → POD)** Headend sends validated ID(s) back to the POD Module. Detailed operational requirements and message protocol/type/syntax are outside the scope of this document. See section 3.2.4.1.
- 10) Host computes its Authentication Key AuthKey_H as described in section 4.1 using SSK_H, the DH public keys DH_pubKey_P and DH_pubKey_H, as well as the Host_ID and POD_ID exchanged in the steps 1 and 2.
- 11) The POD computes its Authentication Key, AuthKey_P, as described in section 4.1 using SSK_P, the DH public keys DH_pubKey_P and DH_pubKey_H, as well as the Host_ID and POD_ID exchanged and validated in the previous messages.
- 12) The POD sends a message to Host to request the Authentication Key AuthKey_H computed by the Host. This request message is implemented by the object CP_data_req() object, as detailed in section 8.3.4.1.
- 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 8.3.4.2.
- 14) Complete Diffie-Hellman protocol to derive the shared DH key (DHKey) in both POD and Host. DHKey is used in the CP key derivation process.

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 45 IAP 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 ID (POD_ID), and Diffie-Hellman public key (DH_pubKey_P) are included in this message.

Table 46 CP_data_req in the IAP 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()	16	2	Defined by and with values set to: Size_indicator = 1 (1 bit, bslbf); Length_field_size = 1 (7 bits, uimsbf); Length_value_byte[0] = 147 (8 bits, bslbf)
CP_system_id	8	1	CP_system_id = 2 (POD CPS)
Send_datatype_nbr	8	1	Send_datatype_nbr shall have the value of 2
For(i=0; i<Send_datatype_nbr; i++) {	(48)	(6)	
Datatype_ID	8	1	When i = 0, Datatype_ID has the value of 6 (APOD_ID);
	8	1	When i = 1, Datatype_ID has the value of 14 (DH_pubKey _P);
Datatype_length	16	2	When i = 0, Datatype_length has the value of 8;
	16	2	When i = 1, Datatype_length has the value of 128;
For (j=0; j<Datatype_length; j++) {	(1088)	(136)	
Data_type	64	8	When i = 0, Data_type = POD_ID;
	1024	128	When i = 1, Data_type = DH_pubKey _P ;
}			
}			
Request_datatype_nbr	8	1	Request_datatype_nbr shall have the value of 2.
For(i=0; i<Request_datatype_nbr; i++)	(16)	(2)	
{			
Datatype_ID	8	1	When i = 0, Datatype_ID has the value of 15 (Cert_Host)
	8	1	When i = 1, Datatype_ID has the value of 13 (DH_pubKey _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 (*Cert_Host*) and Diffie-Hellman public key (*DH_pubKey_H*) are included in this message.

Table 47 CP_data_cnf in the POD CPS 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()	16	2	Defined by and with values set to: Size_indicator = 1 (1 bit, bslbf); Length_field_size = 1 (7 bits, uimsbf); Length_value_byte[0] = 224 (8 bits, bslbf)
CP_system_id	8	1	CP_system_id = 2 (POD CPS)
Send_datatype_nbr	8	1	Send_datatype_nbr shall have the value of 3.
For(i=0; i<Send_datatype_nbr; i++) {	(48)	(6)	
Datatype_ID	8	1	When i = 0, Datatype_ID has the value of 15 (<i>Cert_Host</i>); and
	8	1	When i = 1, Datatype_ID has the value of 13 (<i>DH_pubKey_H</i>);
Datatype_length	16	2	When i = 0, Datatype_length has the value of 88; and
	16	2	When i = 1, Datatype_length has the value of 128.
For (j=0; j<Datatype_length; j++) {	(1728)	(216)	
Data_type	704	88	When i = 0, Data_type = <i>Cert_Host</i> ; and
	1024	128	When i = 1, Data_type = <i>DH_pubKey_H</i>
}			
}			
}			

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 48 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 49 CP_data_req in the Authentication Key Verification Request Message

Message Syntax	bits	bytes	Description
<pre> CP_data_req () { CP_data_req_tag length_field() CP_system_id Send_datatype_nbr Request_datatype_nbr For(i=0; i<Request_datatype_nbr; i++) { Datatype_ID } } </pre>	<p>24</p> <p>8</p> <p>8</p> <p>8</p> <p>8</p> <p>(8)</p> <p>8</p>	<p>3</p> <p>1</p> <p>1</p> <p>1</p> <p>1</p> <p>(1)</p> <p>1</p>	<p>Has the value of 9F9002 (hex)</p> <p>Length_field () is defined in EIA-679-A (Part B) section 7. The length_field() in this message shall have the following values set: size_indicator = 0, length_value = 4</p> <p>CP_system_id = 2</p> <p>Send_datatype_nbr shall have the value of 0.</p> <p>Request_datatype_nbr shall have the value of 1.</p> <p>Datatype_ID has the value of 22 (<i>AuthKeyH</i>, see Table 33).</p>

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 50 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 EIA-679-A (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 (<i>AuthKey_H</i> , see Table 33)
Datatype_length	16	2	Datatype_length has the value of 20 (see Table 33)
For (j=0; j<Datatype_length; j++) {			
Data_type	160	20	Data_type = <i>AuthKey_H</i> (see Table 33)
}			
}			
}			

8.4 Two-way System POD CPS Message Protocol

8.4.1 Protocol Flow Overview

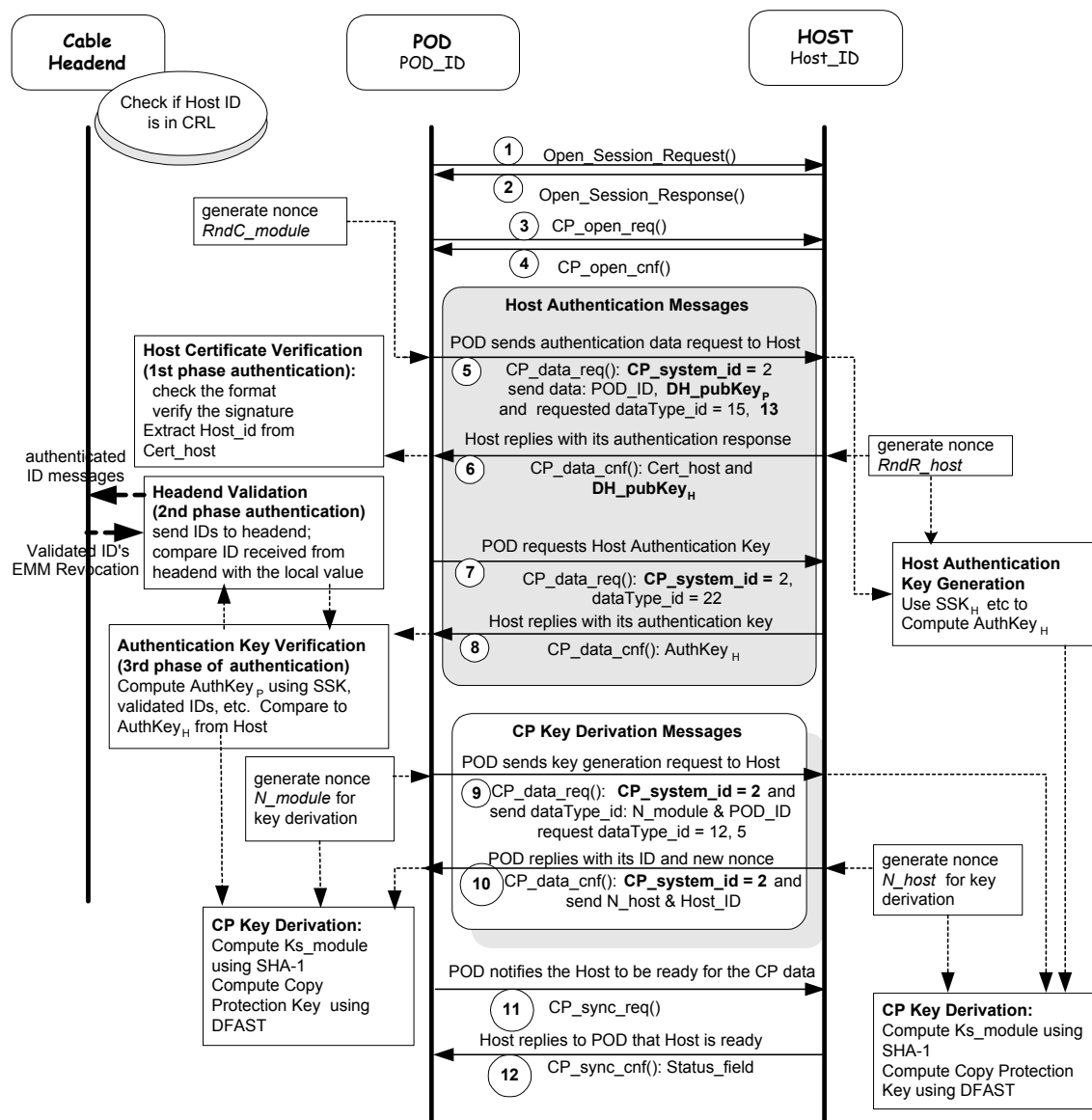


Figure 7 Two-way System POD CPS Message Protocol Overview

Figure 7 gives an overview of the POD CPS Protocol message flow in the two-way system. 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 51 Two-way System POD CPS 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	Host authentication data
6	CP_data_cnf	APDU / 9F9003	Section 8.3.2.2	
7	CP_data_req	APDU / 9F9002	Section 8.3.4.1	Authentication Key verification
8	CP_data_cnf	APDU / 9F9003	Section 8.3.4.2	
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 / 9F9002	Section 8.2.4.1	POD & Host Synchronization
12	CP_sync_cnf	APDU / 9F9003	Section 8.2.4.2	

8.4.2 Host Authentication Protocol Implementation

Similar to the one-way system, the POD CPS Host authentication in a two-way system is achieved in a three-step process. The first step of authentication is based on the certificate signature verification. The second step of authentication is based on headend being able to confirm that the Host ID is not included in CRLs, and the POD being able to confirm the Host ID received from headend is the same as the one stored locally. The third 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.

As with the on-way system, the POD will attempt to perform a re-authentication at power-up if it holds a valid Authentication Key utilizing steps 11 & 12 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 and the POD can proceed with CP key generation. If the keys do not match, then a full Host authentication will be performed, starting with step 1 below. The only difference between a one-way system and a two-way system is that in a one-way system, Host ID and POD ID are reported to the headend via telephone system; in a two-way system, Host ID and POD ID are sent to the headend via a private CA message.

8.4.2.1 Two-way System POD CPS Protocol Steps Full Authentication

- 1) The POD Module initiates the authentication protocol by sending a challenge request to the Host. This challenge request contains its ID (*POD_ID*) and a 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 is detailed in section 8.3.2.1.
- 2) After receiving CP_data_req(), Host sends its reply with its certificate (*Cert_Host*) and 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 Module checks if certificate is valid by:
 - a) Checking the value of the certificate type or format field; and
 - b) Verifying the DSA signature.
- 4) If *Cert_Host* is valid, then the POD extracts *Host_ID* from the certificate.
- 5) **(POD → Cable Headend)** POD sends *Host_ID* and *POD_ID* information to the headend in an authenticated message.
- 6) **(Cable Headend CRL Checking:** second step of authentication) Check if *Host_ID* is in the CRLs. This check may not occur in real time; see section 3.2.4.2.
- 7) **(Cable Headend → POD)** Send EMM to authorize the POD; see section 3.2.4.2.
- 8) **(Cable Headend → POD)** Headend sends validated ID(s) back to the POD Module. Detailed operational requirements and message protocol/type/syntax are outside the scope of this document. See section 3.2.4.2.
- 9) Host computes its Authentication Key *AuthKey_H* by applying the SHA-1 hash function to *SSK_H*, the DH public keys *DH_pubKey_P* and *DH_pubKey_H*, as well as the *Host_ID* and *POD_ID* exchanged in step 1 and 2.
- 10) The POD computes Authentication Key, *AuthKey_P* computed by applying the SHA-1 hash function to *AuthKey_P*, the DH public keys *DH_pubKey_P* and *DH_pubKey_H*, as well as the *Host_ID* and *POD_ID* exchanged and validated in the previous messages.
- 11) The POD sends a message to Host to request the Authentication Key *AuthKey_H* computed by the Host. This request message is implemented by the object CP_data_req() object, as detailed in section 8.3.4.1.
- 12) 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 8.3.4.2.
- 13) The POD and Host complete the Diffie-Hellman protocol to derive the shared DH key (*DHKey*). The *DHKey* is used in the CP key derivation process.

8.5 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, the CCI byte 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 CCI with the fingerprint appended in a data request message. Finally, the Host sends a reply message without a data payload.

Table 52 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 (<i>CCI_N_module</i>) and the program number (<i>program_number</i>), the same one found in the CA_pmt_req() message. (CP_system_id = 2, send datatype_id = 24, 26, and receive 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 generated by the Host (<i>CCI_N_host</i>) and the program number (<i>program_number</i>). (CP_system_id = 2, send datatype_id = 19, 26).	POD ← Host
CP_data_req	9F9002	POD Module sends the CCI payload (<i>CCI_data</i>) and the program number (<i>program_number</i>) and the calculated message digest (<i>CCI_auth</i>). (CP_system_id = 2, send datatype_id = 25, 26, 27).	POD → Host
CP_data_cnf	9F9003	Host replies to POD Module. (CP_system_id = 2)	POD ← Host

Table 53 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 33.
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 (<i>CCI_N_module</i>).
	8	1	i = 1, Datatype_id has the value of 26 (<i>program_number</i>).
Datatype_length	16	2	i = 0, Datatype_length has the value of 0x08.
	16	2	i = 1, Datatype_length has the value of 0x02.
for (j=0; j<Datatype_length; j++)			
{			
Data_type	64	8	When i = 0, Data_type = <i>CCI_N_module</i>
	16	2	When i = 1, Data_type = <i>program_number</i>
}			
}			
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 (<i>CCI_N_host</i>)
	8	1	When i=1, Datatype_id has the value 26 (<i>program_number</i>).
}			
}			

Table 54 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 33.
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 (<i>CCI_N_host</i>).
	8	1	i = 1, Datatype_id has the value of 26 (<i>program_number</i>).
Datatype_length	16	2	i = 0, Datatype_length has the value of 8.
	16	2	i = 1, Datatype_length has the value of 2.
for (j=0; j<Datatype_length; j++)			
{			
Data_type	64	8	When i = 0, Data_type = <i>CCI_N_host</i> .
	16	2	When i = 1, Data_type = <i>program_number</i> .
}			
}			
}			

Table 55 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 34.
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 (<i>CCI_data</i>).
	8	1	i = 1, Datatype_id has the value of 26 (<i>program_number</i>).
	8	1	i = 2, Datatype_id has the value of 27 (<i>CCI_auth</i>).
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 = <i>CCI_data</i> .
	16	2	When i = 1, Data_type = <i>program_number</i> .
	160	20	When i = 1, Data_type = <i>CCI_auth</i> .
}			
}			
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 (<i>CCI_ack</i>).
Datatype_id	8	1	When i=0, Datatype_id has the value 26 (<i>program_number</i>).
}			
}			

Table 56 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 list in Table 34.
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 (<i>CCI_Ack</i>).
	8	1	i=1, Datatype_id has the value of 26 (<i>program_number</i>).
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		20	When i=0, Data_type = <i>CCI_Ack</i> .
		2	When i=1, Data_type = <i>program_number</i> .
}			
}			
}			

Appendix A. Luhn Check Digit

The Luhn check digit is calculated using the following algorithm*.

1. Convert the value into decimal format.
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 2997 2A1F (hexadecimal):

1. Convert to the decimal value 4,992,739,871.
2. Separate this decimal number into odd and even digits starting from the right (least significant digit):

digit #: ¹⁰987654321

'odd' digits: 9, 2, 3, 8, 1

'even' digits: 4, 9, 7, 9, 7

3. Multiply each 'odd' digit by 2:

9, 2, 3, 8, 1 → 18, 4, 6, 16, 2

4. Add the 'even' digits and each individual digit of the products above:

$[4 + 9 + 7 + 9 + 7] + [1 + 8 + 4 + 6 + 1 + 6 + 2] = 64$

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

$10 - 4 = 6$

6. Appended this digit to the right of the decimal ID number for display to subscribers in unidirectional Host validation:

49,927,398,716 (which may be displayed on screen as "00-049-927-398-716")

*Further information was available at <http://staff.semel.fi/~kribe/document/luhn.htm> as of 14 July 2000.

Appendix B. Applying CPKey to DES Engine

Many DES engines take the cryptographic key in 64-bit format as described in FIPS-PUB 46-2 and FIPS-PUB 81. In the meantime, as part of the CPKey refresh cycle a 56-bit integer key (K_{dfast}) is generated.

The 64-bit key is generated from K_{dfast} by adding a parity bit to each 7-bit block. Here we have a K_{dfast} in a 56-bit format:

$$K_{dfast} = K_1 K_2 K_3 \dots K_{56}$$

K_1 is the most significant bit of K_{dfast} . By adding parity bits we get the 64-bit key:

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

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

Here's an example. The original value of CPKey is:

$$\begin{aligned} K_{dfast} &= 0123456789abcd_{(16)} \\ &= 00000001\ 00100011\ 01000101\ 01100111\ 10001001\ 10101011\ 11001101_{(2)} \end{aligned}$$

Break it into eight 7-bit blocks:

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

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

$$\begin{aligned} K_{64bit} &= 00000001\ 10010001\ 11010000\ 10101101\ 01111001\ 01001100\ 10101110\ 10011011_{(2)} \\ &= 0191d0ad794cae9b_{(16)} \end{aligned}$$

Examples of CP encryption on MPEG 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 packets before and after CP encryption respectively (cleartext and encrypted).

Example 1

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

A null packet. CP encryption just leaves the packets that don't belong to a copy protected program.

Example 2

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

A packet without adaptation field that belongs to a copy protected program. DES encryption starts right after the packet header. **transport_scrambling_control** field is changed from **00** to **11** (4th byte: **1c** to **dc**). 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

```

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)

```

A packet with adaptation field that belongs to a copy protected program. DES encryption starts after the adaptation field, which takes 3 bytes for this packet (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 the same way.

(References)

- ◆ FIPS-PUB 81 <<http://www.itl.nist.gov/fipspubs/fip81.htm>>
- ◆ FIPS-PUB 46-2 <<http://www.itl.nist.gov/fipspubs/fip46-2.htm>>