Digital Transmission Content Protection 2 (DTCP2) Specification Volume 1 (Informational Version)

Intel Corporation
Maxell, Ltd.
Panasonic Holdings Corporation
Sony Group Corporation
Toshiba Corporation

Revision 1.0.4
October 25, 2022

Preface

Notice

THIS DOCUMENT IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE. Intel, Maxell, Ltd., Panasonic, Sony, and Toshiba (collectively, the "5C") disclaim all liability, including liability for infringement of any proprietary rights, relating to use of information in this Specification. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted herein.

Some portions of this document, identified as "Draft" are in an intermediate draft form and are subject to change without notice. Adopters and other users of this Specification are cautioned these portions are preliminary, and that products based on it may not be interoperable with the final version or subsequent versions thereof.

Intellectual Property

Implementation of this Specification requires a license from the Digital Transmission Licensing Administrator.

Contact Information

Feedback on this Specification should be addressed to dtcp-services@dtcp.com.

Printing History:

2017-06-07	**Digital Transmission Content Protection (DTCP2) Specification Volume 1 Revision 1.0 (Informational Version)**
2018-01-10	**Digital Transmission Content Protection (DTCP2) Specification Volume 1 Revision 1.0.1 (Informational Version)**
2018-09-12	**Digital Transmission Content Protection (DTCP2) Specification Volume 1 Revision 1.0.2 (Informational Version)**
2019-07-11	**Digital Transmission Content Protection (DTCP2) Specification Volume 1 Revision 1.0.3 (Informational Version)**
2022-10-25	**Digital Transmission Content Protection (DTCP2) Specification Volume 1 Revision 1.0.4 (Informational Version)**

TABLE OF CONTENTS

1 Introduction

1.1 Purpose and Scope

The Digital Transmission Content Protection 2 (DTCP2) Specification defines a cryptographic protocol for protecting audio/video entertainment content from unauthorized copying, intercepting, and tampering as it traverses digital interconnects. Only commercial entertainment is protected by this system.

The use of this Specification and access to the intellectual property and cryptographic materials required to implement it will be the subject of a license. The Digital Transmission Licensing Administrator (DTLA) is responsible for establishing and administering the content protection system described in this Specification.

1.2 Manufacture of Compliant Devices

In addition to the DTCP2 Specification all compliant devices must meet all content protection requirements of DTCP2 set forth in the DTCP2 Specification, Compliance Rules, and Robustness Rules.

1.3 Overview

Device authentication and key exchange (AKE)

Before sharing keys and other valuable information, a connected device must first verify that another connected device is authentic.

Content encryption

All commercial entertainment content is encrypted using AES-128 encryption.

Content Management Information (CMI)

Content owners need a way to specify how their content can be used ("copy-one-generation," "copy-never," etc.). This content protection system is capable of securely communicating copy control information (CCI) between devices using the CMI descriptor.

System renewability

Devices receive and process System Renewability Messages (SRMs) created by the DTLA and distributed with content and new devices. System renewability ensures long-term integrity of the system through the revocation of compromised devices.

1.4 References

This Specification shall be used in conjunction with the following publications. When the publications are superseded by an approved revision, the revision shall apply.

IEEE 1363-2000, IEEE Standard Specification for Public-Key Cryptography

IEEE 1363a-2004, IEEE Standard Specifications for Public-Key Cryptography - Amendment 1: Additional Techniques

NIST, Secure Hash Standard, FIPS PUB 180-4, March 2012

NIST, Digital Signature Standard (DSS), FIPS PUB 186-4 Digital Signature Standard (DSS) July 2013

NIST, Advanced Encryption Standard (AES), FIPS PUB 197, November 26, 2001

NIST, The Keyed-Hash Message Authentication Code (HMAC), FIPS PUB 198-1, July 2008

NIST, Recommendation for Block Cipher Modes of Operation, SP 800-38A, December 2001

NIST, Recommendations for Key Derivation Using Pseudorandom Functions (Revised), SP 800-108, October 2009

NIST, Recommended ECC Elliptic Curves for Federal Government Use, July 1999

NIST, Recommendation for Random Number Generation Using Deterministic Random Bit Generators, SP 800-90A Revision 1, June 2015

RFC768 User Datagram Protocol

RFC791 Internet Protocol

RFC793 Transmission Control Protocol

RFC1945 Hypertext Transfer Protocol – HTTP/1.0

RFC2616 Hypertext Transfer Protocol – HTTP/1.1

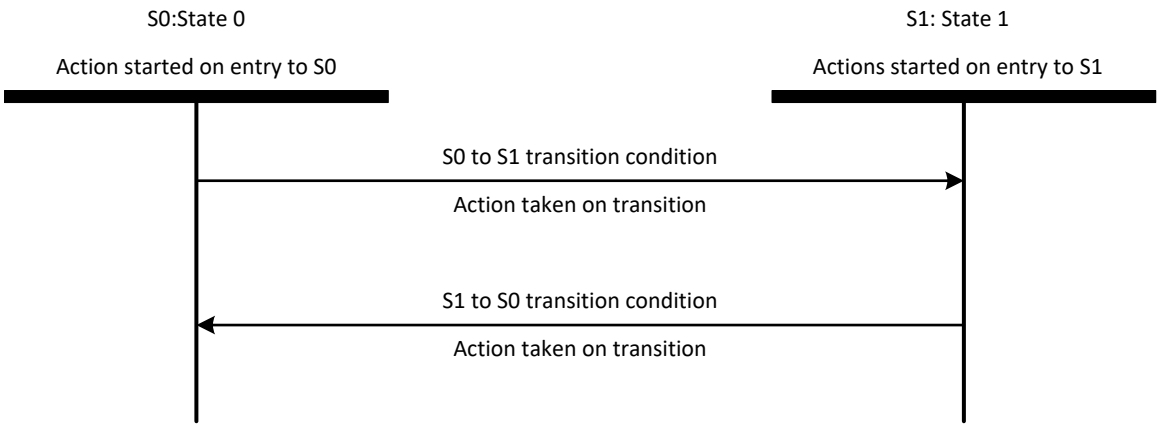
RFC1889 RTP: A Transport Protocol for Real-Time Applications

Macrovision Copy Protection Process for DVD Products, Revision 7.1.D1, September 30, 1999

ContentDirectory:2 Service Template Version 1.01, UPnP Forum, May 31, 2006

1.5 State Machine Notation

State machines are employed throughout this document to show various states of operation. These state machines use the style shown in the following figure.



State machines make three assumptions:

- Time elapses only within discrete states.
- State transitions are instantaneous, so the only actions taken during a transition are setting flags and variables and sending signals.
- Every time a state is entered, the actions of that state are started. A transaction that points back to the same state will restart the actions from the beginning.

1.6 Notation

The following notation will be used:

$[X]_{msb\_z}$ The most significant z bits of X

$[X]_{lsb\_z}$ The least significant z bits of X

$S_{X^{-1}}[M]$ Sign M using EC-DSA with private key X^{-1}

X || Y Ordered Concatenation of X with Y.

$X \oplus Y$ Bit-wise Exclusive-OR (XOR) of two strings X and Y.

1 KB = 1024 Bytes

1 MB = 1024 x 1024 Bytes

1.7 Numerical Values

Three different representations of number are used in this Specification. Decimal numbers are represented without any special notation. Binary number are represented as a string of binary (0, 1) digits followed by a subscript 2 (e.g., 1010_2). Hexadecimal numbers are represented as a string of hexadecimal digits 0..9 , A..F followed by a subscript 16 (e.g., $3C2_{16}$).

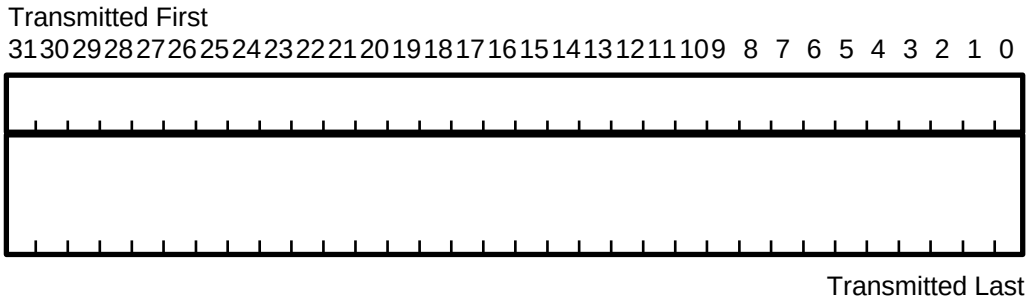
1.8 Byte Bit Order

Data is depicted from most significant to least significant when scanning document from top to bottom and left to right.

7	6	5	4	3	2	1	0
msb			(MSB)				
				(LSB)			lsb

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Msb																															
																															lsb

1.9 Packet Transmission Format



1.10 Terminology

This section contains definitions of terms used in this Specification for which the definitions could not easily be obtained from other sources.

Bound Copy

Copy stored in accordance with the Compliance Rule specified in 2.2.1.2 of Exhibit B, Part 1-B of the DTCP2 Adopter Agreement.

Reserved

The term "Reserved", when used to define the syntax of the data structure, indicates that the field may be used for future extensions. All the bits of reserved field in the syntax of a data structure shall be set to 0_2 unless otherwise noted. The term "Reserved", when used to define the meaning of values, indicates that the reserved values may be used for future extensions. The reserved values shall never be used in this version.

1.11 Abbreviations

The following is an alphabetical list of abbreviations and acronyms used throughout this document.

Advanced Encryption Standard (AES)
Analog Protection System (APS)
Authentication and Key Exchange (AKE)
Automatic Gain Control (AGC)

Certification Authority (CA)
Certificate Revocation List (CRL)
Copy Control Information (CCI)
Copy Generation Management System (CGMS)
Content Management Information (CMI)
Cipher-Block-Chaining (CBC)

Diffie-Hellman (DH)
Digital Signature Algorithm (DSA)
Digital Signature Standard (DSS)
Digital Transmission Content Protection (DTCP)
Digital Transmission Licensing Administrator (DTLA)
Digital Video Disc (DVD)
Discrete Logarithm Signature Primitive, DSA version (DLSP-DSA)
Discrete Logarithm Verification Primitive, DSA version (DLVP-DSA)
Socket used for AKE commands (DTCP2 Socket)

Encryption Plus Non-assertion (EPN)
Elliptic Curve (EC)
Elliptic Curve Cryptography (ECC)
Elliptic Curve Digital Signature Algorithm (EC-DSA)
Elliptic Curve Diffie-Hellman (EC-DH)
Elliptic Curve Secret Value Derivation Primitive, Diffie-Hellman version (ECSVDP-DH)
Elliptic Curve Signature Schemes with Appendix (ECSSA)

Federal Information Processing Standards (FIPS)

Hash-based Message Authentication Code (HMAC)
Hypertext Transfer Protocol (HTTP)

Institute of Electrical and Electronics Engineers (IEEE)
International Electrotechnical Commission (IEC)
International Organization for Standardization (ISO)
Internet Protocol (IP)

Least Significant Bit (lsb)
Least Significant Byte (LSB)

Message Authentication Code (MAC)
Most Significant Bit (msb)
Most Significant Byte (MSB)

National Institute of Standards and Technology (NIST)

Protected Content Packet (PCP)
Public Key Infrastructure (PKI)

Random Number Generator (RNG)
Real-time Transfer Protocol (RTP)
Remote Access (RA)
Remote Access Connection Counter (RACC)
Round Trip Time (RTT)

Secure Hash Algorithm (SHA)
Secure Hash Standard (SHS)
IP-address concatenated with port number [e.g. host | port] (Socket)
System Renewability Message (SRM)

Time To Live (TTL)
Transmission Control Protocol (TCP)

User Datagram Protocol (UDP)

2 DTCP2 System

DTCP2 system consists of 2 entities a source device with a Source Function and a sink device with a Sink Function. There may be a device with both a Source Function and a Sink Function, which behaves as not only a source device but also a sink device, such as Bridge Device, while such device shall follows the rules for not only Sink Function/sink device but also Source Function/source device unless otherwise noted.

2.1 Source Device

Source devices upon power up are initialized and transition to an unauthenticated state and await the successful completion of the Authentication and Key Exchange (AKE) to achieve the Authenticated state.

2.2 Sink Device

Sink devices upon power up are initialized and transition to an unauthenticated state and await the successful completion of the Authentication and Key Exchange (AKE) to achieve the Authenticated state.

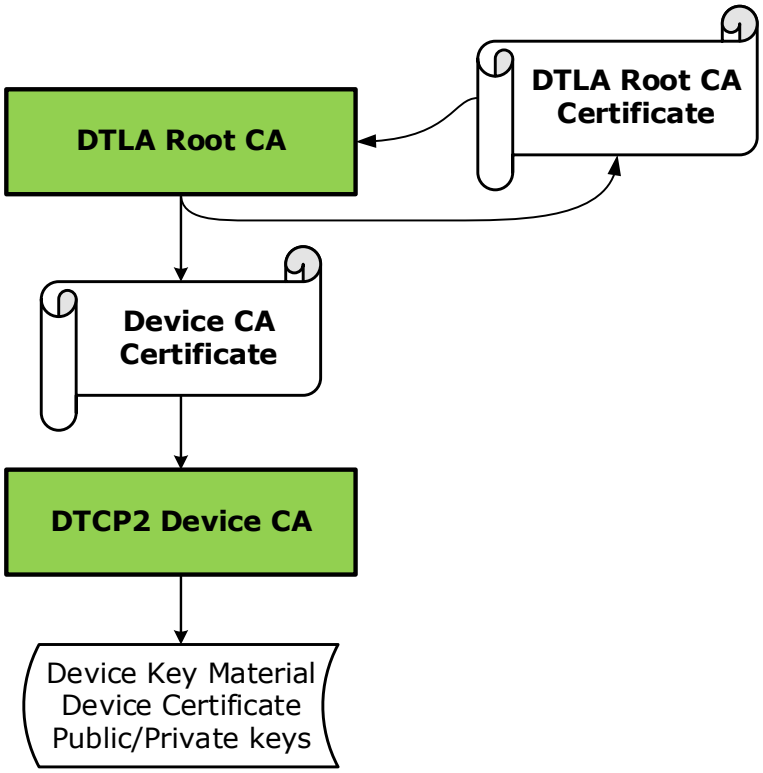
2.3 Bridge Device

Bridge Device has both a Sink Function and a Source Function where the Sink Function decrypts content received from an upstream source device and the Source Function simultaneously encrypts and transmits such decrypted content to one or more downstream sink devices. Bridge Device behaves as a sink device to upstream source devices and also behaves as a source device to downstream sink devices.

3 Public Key Infrastructure (PKI)

DTCP2 is an ECC based PKI using the NIST P-256 ECC domain parameters (FIPS PUB 186-4). The DTCP2 PKI consists of a DTLA Root Certification Authority (CA) and a DTCP2 Device CA. The DTCP2 Device CA generates the DTCP2 device keying material; device private key and the corresponding device certificate which contains the device public key.

The DTCP2 Keying Material Order Guide (KMOG) provided to Adopters specifies the DTCP2 device keying material delivery format and contents.



3.1 DTCP2 Certificate Formats

3.1.1 DTLA Root CA Certificate

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Certificate Type				Reserved																				DTLA Root CA ID							
DTLA Root CA ID continued (Total 40 bits)																															
DTLA Root CA EC-DSA Public Key (512 bits)																															
EC-DSA signature of all preceding fields signed by DTLA Root CA (512 bits for c and followed by d value)																															

The DTLA Root CA Certificate is comprised of the following fields:

- **Certificate Type** (4 bits): The Value of 8 shall indicate the DTLA Root CA Certificate
- **Reserved** (20 bits)
- **DTLA Root CA Id**entifier (40 bits)
- **DTLA Root CA Public Key** (512 bits)
- **EC-DSA Signature** self-signed using DTLA Root CA Private Key (512 bits)

3.1.2 DTCP2 Device CA Certificate

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Certificate Type				Reserved																				DTCP2 Device CA ID							
DTCP2 Device CA ID continued (Total 40 bits)																															
DTCP2 Device CA EC-DSA Public Key (512 bits)																															
EC-DSA signature of all preceding fields signed by DTLA Root CA (512 bits for c and followed by d value)																															

The DTCP2 Device CA Certificate is comprised of the following fields:

- **Certificate Type** (4 bits): The value of 9 shall indicate DTCP2 Device CA Certificate
- **Reserved** (20 bits)
- **DTCP2 Device CA Id**entifier (40 bits)
- **DTCP2 Device CA Public Key** (512 bits)
- **EC-DSA Signature** signed using DTLA Root CA Private Key (512 bits)

3.1.3 DTCP2 Device Certificate

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Certificate Type				Format				Dev Gen				Reserved (zero)								CK	RSV		AP	DTCP2 Device ID							
DTCP2 Device ID continued (Total 40 bits)																															
DTCP2 Device EC-DSA Public Key (512 bits)																															
EC-DSA signature of all preceding fields signed by DTCP2 Device CA (512 bits for c and followed by d value)																															

The DTCP2 Device Certificate is comprised of the following fields:

- **Certificate Type** (4 bits): The value of 10 (A_{16}) shall indicate DTCP2 Device Certificate
- **Format** (4 bits): This field specifies certificate format and currently only one format is defined:
  - 1_{16} : DTCP2 Device Certificate Format 1
- **Dev**ice **Gen**eration (4 bits): This field (X_{SRMG}) indicates the non-volatile memory capacity for SRM and therefore the maximum generation of SRM this device supports (See Section 8). The encoding 0 indicates that the device shall store a First-Generation SRM and the encoding 1 indicates that the device shall store a Second-Generation SRM, and so on.
- **Reserved** (8 bits)
- **CK flag** (1 bit): A value of zero indicates the device is using unique DTCP2 keying material (Unique-key Device) and a value of one indicates that the device is using common keying material (Common-key Device).
- **RSV** (2 bits): Reserved and set to zero.
- **AP flag**[1] (1 bit): Authentication Proxy flag. A device certificate with an AP flag value of one is used by a Bridge Device, which receives a content stream using a Sink Function that streams to another interface using a Source Function. The procedures for processing this field are specified in Section 9.
- **DTCP2 Device Id**entifier (40 bits)
- **DTCP2 Device Public Key** (512 bits)
- **EC-DSA Signature** signed using DTCP2 Device CA Private Key (512 bits)

[1] AP flag is zero when device using common keying material does not have a DTCP2 bridge function unlike DTCP1.

4 Authentication and Key Exchange (AKE)

4.1 Introduction

AKE employs the public key based Elliptic Curve Digital Signature Algorithm (EC-DSA) for signing and verification. It also employs the Elliptic Curve Diffie-Hellman (EC-DH) key exchange algorithm to generate a shared authentication key.

The notation introduced in this section is used to describe the cryptographic processes. All operations in the elliptic curve domain are calculated on an elliptic curve E defined over GF(p).

4.2 Description and Lengths of Cryptographic Elements

4.2.1 Domain Parameter and CA Key Lengths

	Description	Size (bits)
p	A prime number greater than 3 of finite field *GF(p)*	256
a, b	The coefficients of elliptic curve polynomial	256 each
G	The basepoint for the elliptic curve *E*	512
r	The order of basepoint *G*	256
L^{-1}	DTCP2 Device CA Private Key of EC-DSA key pair which is an integer in the range (*1*, *r–1*)	256
L^{1}	DTCP2 Device CA Public Key of EC-DSA key pair where $L^1 = L^{-1}G$	512
C^{-1}	DTLA Root CA Private Key of EC-DSA key pair which is an integer in the range (*1*, *r–1*)	256
C^{1}	DTLA Root CA Public Key of EC-DSA key pair where $C^1 = C^{-1}G$	512

4.2.2 Device Private/Public Key Lengths

	Description	Size (bits)
X^{-1}	Device Private Key of EC-DSA key pair which is an integer in the range (*1*, *r–1*)	256
X^{1}	Device Public Key of EC-DSA key pair where $X^1 = X^{-1}G$	512

4.2.3 Variables used during Authentication

The following additional values are generated and used by the devices during Authentication:

	Description	Size (bits)
X_n	Nonce (random challenge generated by RNG)	128
X_K	Random value used in EC-DH key exchange generated by RNG in the device (integer in the range [*1*, *r-1*])	256
X_V	EC-DH first phase value (X_KG) calculated in the device (point on the elliptic curve *E*)	512
X_{SRMV}	Version number of the system renewability message (SRMV) stored by the device (See Section 8)	16
X_{SRMC}	This value plus one indicates the number of SRM parts (generations) currently stored by the device (See Section 8). The value of 0 means the First Generation part is stored, and the value of 1 means the First and Second Generation parts are stored.	4
K_{AUTH}	Authentication key made by shared data created through EC-DH key exchange	128

4.2.4 Implementation ID

The Implementation ID is a 40 bit field sent in the RESPONSE command during AKE and is made up of two fields as shown below:

Implementation ID	
Adopter ID (24 bits)	Implementation Number (16 bits)

4.3 Cryptographic Functions

4.3.1 Hash Function

SHA-256, as described in Secure Hash Standard, FIPS PUB 180-4, March 2012 is the algorithm used in DSS to generate a message digest of length 256 bits.

4.3.2 Random Number Generator (RNG)

A device may use a true random number generator based on the measurement of physical phenomena. If a device uses a pseudorandom number generator, it shall meet or exceed NIST SP800-90A Rev1 standard.

4.3.3 Elliptic Curve Cryptography (ECC)

These cryptographic algorithms are based upon cryptographic schemes, primitives, and encoding methods described in IEEE 1363-2000 and IEEE 1363a-2004.

An Elliptic Curve Cryptosystem (ECC) is used as the cryptographic basis for DH and DSA.

The definition field classifies ECC implementations. For this system, the definition field used is *GF(p)* where *p* is a large prime number greater than three. An elliptic curve *E* over the field *GF(p)*, where $p > 3$, is defined by the parameters a and b and the set of solutions *(x, y)* to the elliptic curve equation together with an extra point often called the point at infinity. The point at infinity is the identity element of the Abelian group, *(E, +)*. The elliptic curve equation used is

$$y^2 = x^3 + ax + b \text{ where } 4a^3 + 27b^2 \neq 0,$$

Where *a, b, x, y*, are elements of *GF(p)*. A point P on the elliptic curve consists of the x-coordinate and the y-coordinate of a solution to this equation, or the point at infinity, and is designated $P = (x_p, y_p)$.

For EC-DSA and EC-DH, a basepoint *G* on the elliptic curve is selected. All operations in the elliptic curve domain are calculated on an elliptic curve *E* defined over *GF(p)*. The public key Y^1 (a point on the elliptic curve) and private key Y^{-1} (a scalar value satisfying $0 < Y^{-1} < r$) for each entity satisfies the equation:

$$Y^1 = Y^{-1} G$$

In specifying the elliptic curve used:

The order of basepoint *G* will have a large prime factor.

4.3.4 Elliptic Curve Digital Signature Algorithm (EC-DSA)

4.3.4.1 Signature

The following signature algorithm is based on the ECSSA digital signature scheme using the DLSP-DSA signature primitive and EMSA1 encoding method defined in of IEEE 1363-2000 and IEEE 1363a-2004.

Input:

- M = the data to be signed
- X^{-1} = the private key of the signing device (must be kept secret)
- p, a, b, G, and r = the elliptic curve parameters associated with X^{-1}

Output:

- $[M]$ = a 512-bit signature of the data, M, based on the private key, X^{-1}

Algorithm:

Step 1, Generate a random value, u, satisfying $0 < u < r$, using RNG. A new value for u is generated for every signature and shall be unpredictable to an adversary for every signature computation. Also, calculate the elliptic curve point, $V = uG$.

Step 2, Calculate $c = x_V \bmod r$ (the x-coordinate of V reduced modulo r). If c = 0, then go to **Step 1**.

Step 3, Calculate $f = [SHA\text{-}256(M)]_{msb\_bits\_in\_r}$. That is, calculate the SHA-256 hash of M and then take the most significant bits of the message digest that is the same number of bits as the size of r.

Step 4, Calculate $d = [u^{-1}(f + cX^{-1})] \bmod r$ (note that u^{-1} is the modular inverse of u mod r while X^{-1} is the private key of the signing device). If d = 0, then go to **Step 1**.

Step 5, Set first 256 bits of $S_{X^{-1}}[M]$ equal to the big endian representation of c, and the second 256 bits of $S_{X^{-1}}[M]$ equal to the big endian representation of d. ($S_{X^{-1}}[M] = c \,||\, d$)

4.3.4.2 Verification

The following verification algorithm is based on the ECSSA digital signature scheme using the DLVP-DSA signature primitive and EMSA1 encoding method defined in of IEEE 1363-2000 and IEEE 1363a-2004.

Input:

- $S_X^{-1}[M]$ = an alleged 512-bit signature (c || d) of the data, M, based on the private key, X^{-1}
- M = the data associated with the signature
- X^1 = the public key of the signing device
- p, a, b, G, and r = the elliptic curve parameters associated with X^{-1}

Output:

- "valid" or "invalid", indicating whether the alleged signature is determined to be valid or invalid, respectively

Algorithm:

Step 1, Set c equal to the first 256 bits of $S_X^{-1}[M]$ interpreted as in big endian representation, and d equal to the second 256 bits of $S_X^{-1}[M]$ interpreted as in big endian representation. If c is not in the range $[1, r-1]$ or d is not in the range $[1, r-1]$, then output "invalid" and stop.

Step 2, Calculate $f = [SHA\text{-}256(M)]_{msb\_bits\_in\_r}$. That is, calculate the SHA-256 hash of M and then take the most significant bits of the message digest that is the same number of bits as the size of r.

Step 3, Calculate $h = d^{-1} \bmod r$, $h_1 = (fh) \bmod r$, and $h_2 = (ch) \bmod r$.

Step 4, Calculate the elliptic curve point $P = (x_P, y_P) = h_1G + h_2X^1$. If P equals the elliptic curve point at infinity, then output "invalid" and stop.

Step 5, Calculate $c' = x_P \bmod r$. If $c' = c$, then output "valid"; otherwise, output "invalid."

4.3.5 Elliptic Curve Diffie-Hellman (EC-DH)

The following shared secret derivation algorithm is based on the ECSVDP-DH primitive defined in IEEE 1363-2000 and IEEE 1363a-2004.

Input:

- Y_V = the Diffie-Hellman first phase value generated by the other device (an elliptic curve point)
- p, a, b, G, and r = the elliptic curve parameters associated with X^{-1}

Output:

- X_V = the Diffie-Hellman first phase value (an elliptic curve point)
- the x-coordinate of X_KY_V = the shared secret generated by this algorithm (must be kept secret from third parties)

Algorithm:

Step 1, Generate a random integer, X_K, in the range [1, r-1] using RNG. A new value for X_K is generated for every shared secret and shall be unpredictable to an adversary. Also, calculate the elliptic curve point, $X_V = X_KG$.

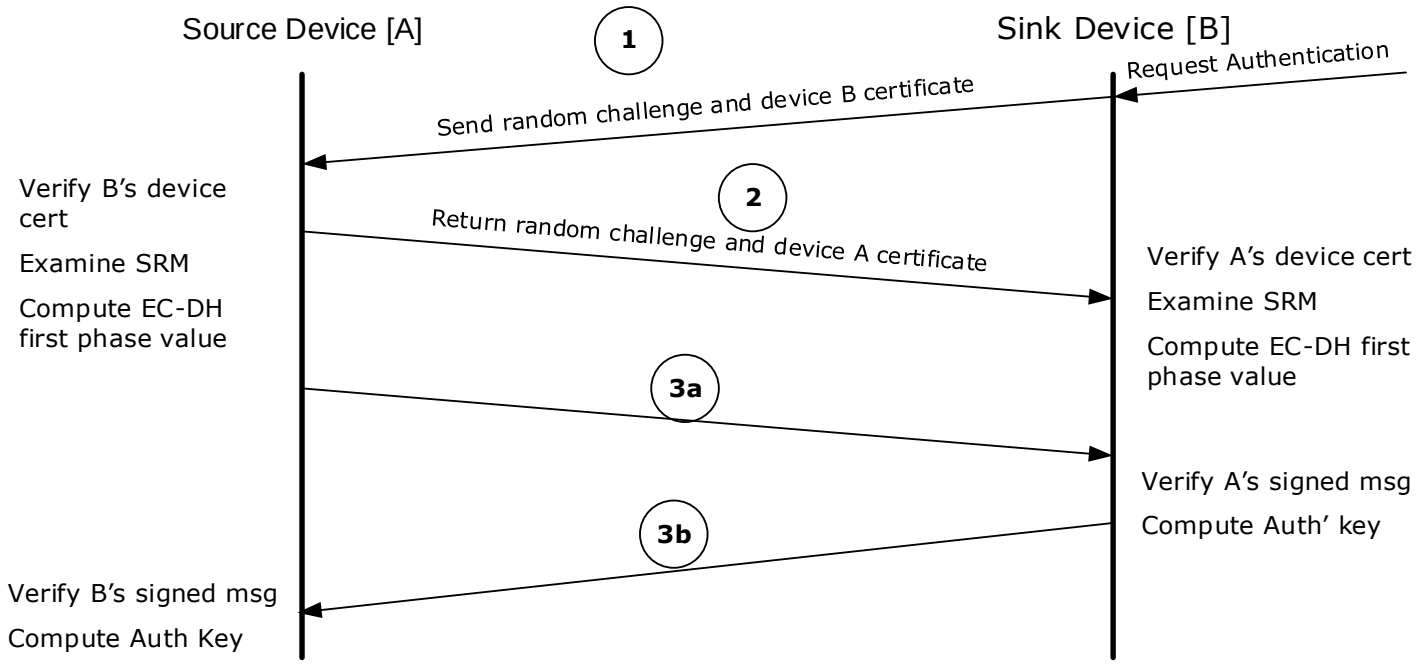
Step 2, Output X_V.

Step 3, Calculate X_KY_V. Output the x-coordinate of X_KY_V as the secret shared.

4.4 Protocol Flow

4.4.1 Protocol Flow Overview

The following Figure gives an overview of the Authentication protocol flow.



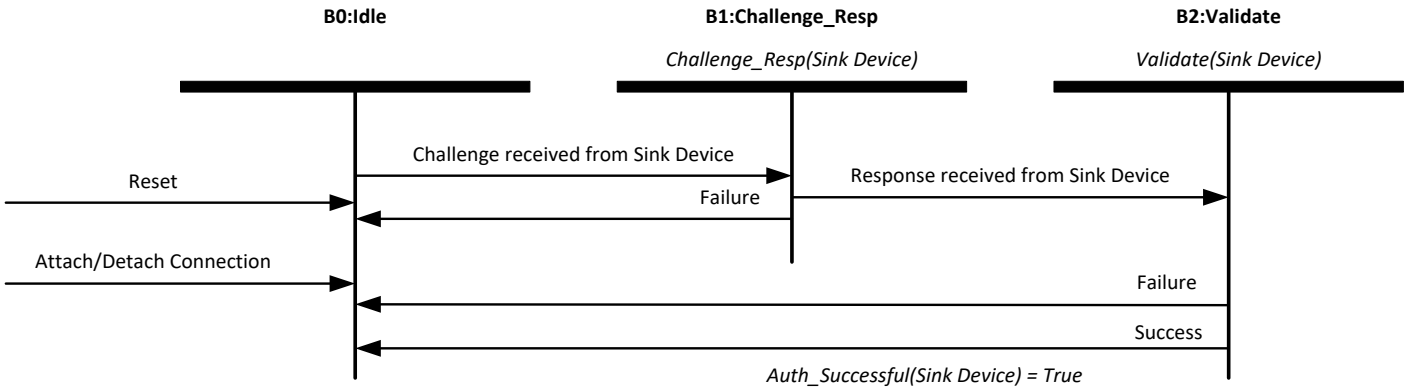
During Authentication:

1. The sink device B requests authentication by sending a random challenge and its device certificate. This can be the result of the sink device attempting to access a protected content stream (whose CCI is set to "Copy-Never," "No-More-Copies," or "Copy-One-Generation", or CCI is "Copy-Freely" and EPN is asserted). The sink device may request authentication on a speculative basis, before attempting to access a content stream.
2. The source device A then returns a random challenge and its device certificate. If the value of the other device's certificate type or format fields is reserved, the authentication should be immediately aborted. After the random challenge and device certificate exchange, each device verifies the integrity of the other device's certificate using EC-DSA. If the DTCP2 Device CA signature is determined to be valid, the devices examine the certificate revocation list embedded in their System Renewability Messages (see Section 8) to verify that the other device has not been revoked. If the other device has not been revoked, each device calculates an EC-DH key exchange first-phase value (see Section 4.3.5).
3. The devices then exchange messages containing the EC-DH key exchange first-phase value, the Renewability Message Version Number and Generation of the System Renewability Message stored by the device, and a message signature containing the other device's random challenge concatenated to the preceding components. The devices verify the signed messages received by checking the message signature using EC-DSA with the other device's public key. This verifies that the message has not been tampered with. If the signature cannot be verified, the device refuses to continue. In addition, by comparing the exchanged version numbers, devices can at a later time invoke the system renewability mechanisms (see Section 8.1.4 for the details of this procedure).

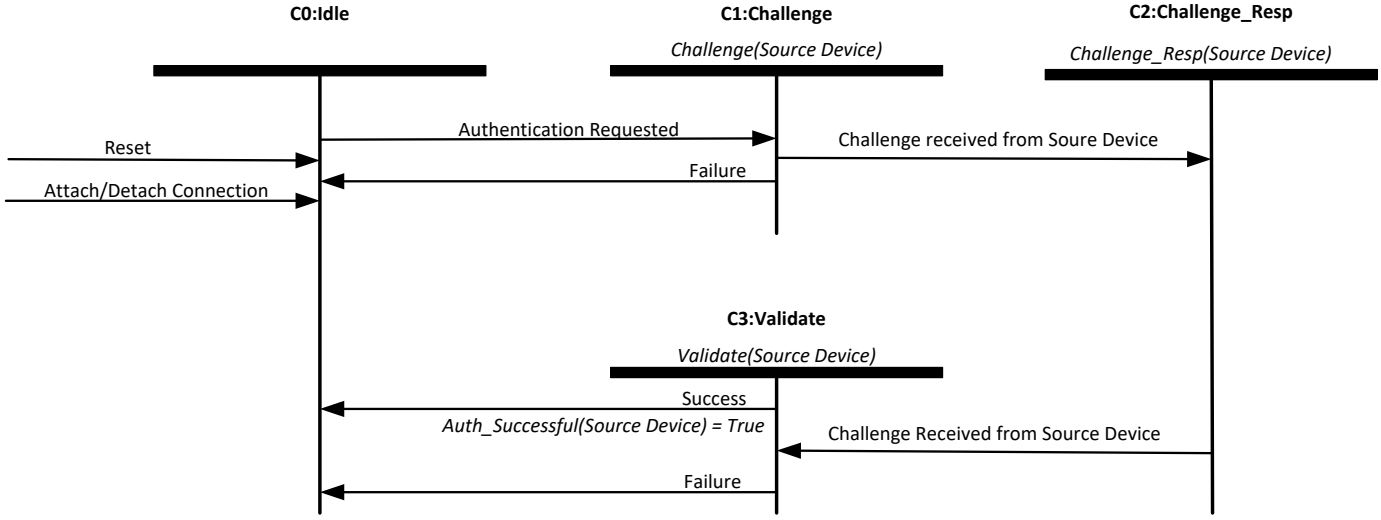
Each device calculates an authentication key by completing the EC-DH key exchange.

4.5 Authentication State Machine

The following Figure shows the *Auth(Device)* state machine from the point of view of a source device receiving a challenge request from a sink device. The detail of this state machine including the function to calculate K_{AUTH} is described in the DTCP2 Specification available under license from the DTLA.



The following figure shows the *Auth(Device)* state machine from the point of view of a **sink device** receiving an Authentication request from a source device. The detail of this state machine including the function to calculate K'_{AUTH} is described in the DTCP2 Specification available under license from the DTLA.



4.6 Localization

Source and sink devices must implement Localization as specified in this section. Note that Remote Access AKE (RA-AKE) is not required to implement Localization. This section specifies requirements for localization and the authentication protocol including proximity check by RTT measurement based on the protocol flow in Section 4.4.

4.6.1 Round Trip Time (RTT) Requirement

Source devices when conducting an AKE with a sink device must perform a RTT test if the sink device's Device ID is not on the source device's RTT registry except for Remote Access AKE (RA-AKE).

Source devices will add a sink device's Device ID to the source device's RTT registry, set the content transmission counter for the sink device to 40 hours or less, and provide an Exchange Key only if the source device measures a RTT value of 7 milliseconds or less during RTT test.

Source devices when transmitting content will decrement content transmission counters of all RTT registered sink devices and are required to remove the Device ID of a sink device from the RTT registry when the corresponding content transmission counter becomes zero.

If the value of CK flag is one in the device certificate of sink device, ID_U in the RESPONSE subfunction from the sink device shall be used instead of Device ID in above processes.

The protocols for the above are specified in Section 4.6.4 and 4.6.5.

4.6.2 Internet Datagram Time To Live (TTL) Requirement

TTL is described in RFC791 and the following requirements only apply to IP datagrams that transport AKE commands described in Section 7. Transmitting devices shall set TTL value of such transmitted IP datagrams to a value no greater than 3 and correspondingly receiving devices shall discard such received IP datagrams which have a TTL value greater than 3 except for the following cases:

- Transmission and reception of IP datagrams required for RA-AKE (including CONT_KEY_CONF subfunction using Remote Exchange Key (K_R)).
- Transmission and reception of IP datagrams for RA_MANAGEMENT subfunction data.
- Transmission and reception of IP datagrams for the Move Protocol between a source device and sink device sharing a Remote Exchange Key (K_R).
- Under the conditions permitted in the DTCP2 Adopter Agreement, Exhibit B "TTL Exception" section.

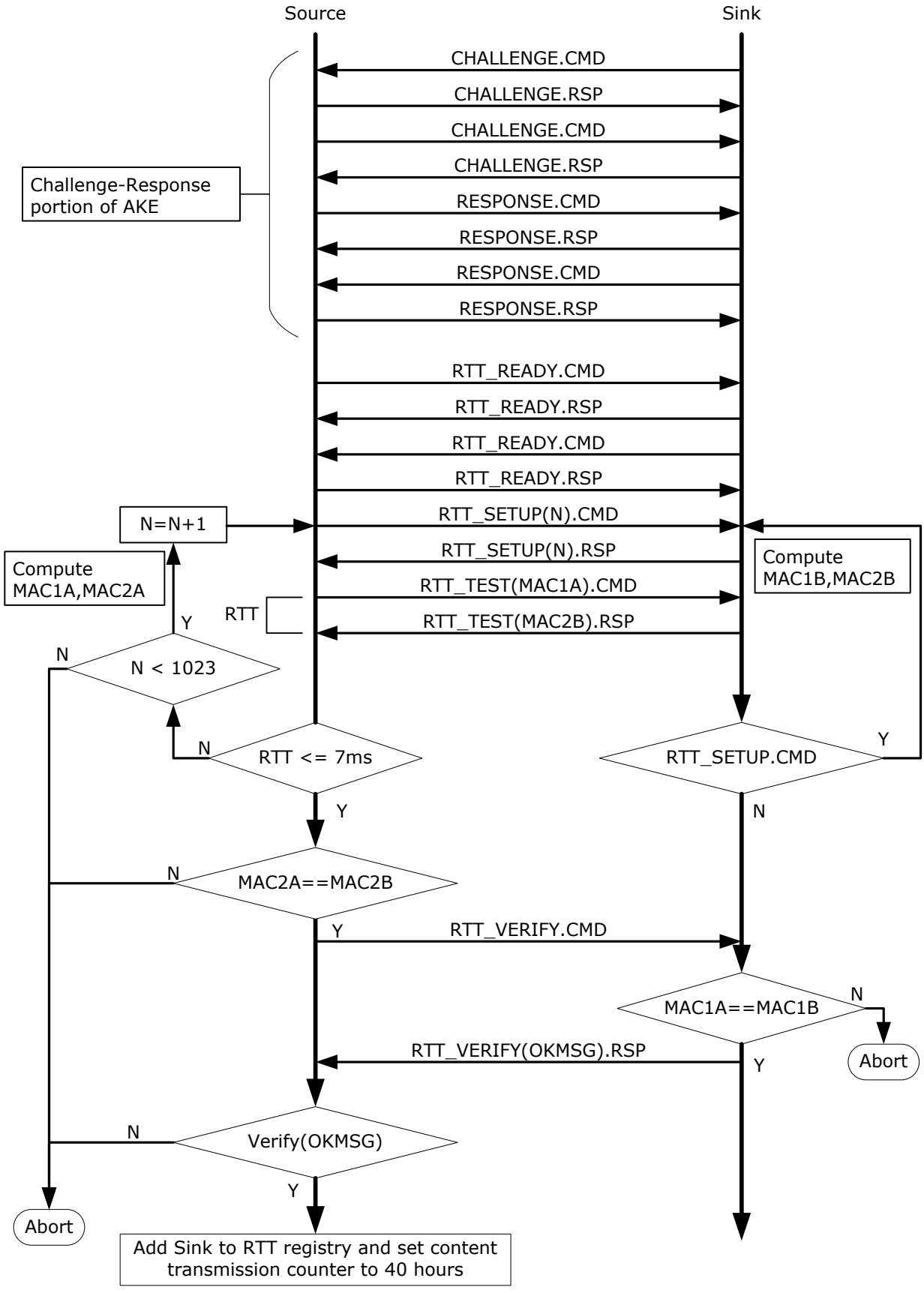
4.6.3 Wireless LAN Security Requirement

DTCP2 devices with integrated Wireless LAN based on IEEE 802.11 standards shall ensure that Wireless LAN security is enabled (e.g. WPA or WPA2, WPA2/WPA mixed) prior to exchanging AKE commands and protected content via such a network interface except for the conditions permitted in the DTCP2 Adopter Agreement, Exhibit B, "Wireless LAN Security Exception" sections.

4.6.4 Protected RTT Protocol

The protected RTT protocol is described in the following figure and is used in RTT-AKE specified in the next section. The RTT protocol is executed after the Challenge-Response portion of the AKE specified in Section 4.4 is completed. SHA-256 is used to construct the messages exchanged during RTT testing protocol to ensure that source and sink devices which completed Challenge-Response portion of AKE are only ones involved in RTT testing. The functions to calculate messages of MAC1A, MAC1B, MAC2A, MAC2B and OKMSG based on the secret value shared in the AKE are described in the DTCP2 Specification available under license from the DTLA.

The following figure includes AKE commands and responses defined in Section 7 where commands are denoted with ".CMD" and responses are denoted with ".RSP".



The RTT_READY command is used to indicate that authentication computation is complete and that source and sink devices are ready to execute the RTT test procedure.

The RTT procedure begins by first establishing value of N using the RTT_SETUP command. N is initially set to zero and can range from 0 to 1023 as maximum permitted RTT trials per AKE is 1024.

After preparation of MAC values corresponding to N, source device will then measure RTT which is the time interval starting after source transmits RTT_TEST command and terminates upon reception of RTT_TEST ACCEPTED response.

If the RTT is greater than 7 milliseconds and the value of N is less than 1023 the source will repeat RTT procedure by incrementing N by 1 and reissue RTT_SETUP and RTT_TEST commands.

If the measured RTT is less than or equal to 7 milliseconds:

The source device compares most recently computed MAC2A to most recently received MAC2B and if not equal the source device aborts RTT procedure else if equal it sends RTT_VERIFY command to sink device.

The sink device will after receipt of RTT_VERIFY command compare the most recently received MAC1A and most recently computed MAC1B and if not equal aborts RTT procedure else if equal it will send OKMSG in RTT_VERIFY ACCEPTED response.

The source device will verify OKMSG and if it is not correct the source device aborts RTT procedure else it will add sink device's Device ID to RTT registry and set content transmission counter to 40 hours or less. If the CK flag is set to 1 in the sink device's certificate, ID_U shall be added to RTT registry instead of Device ID in above process.

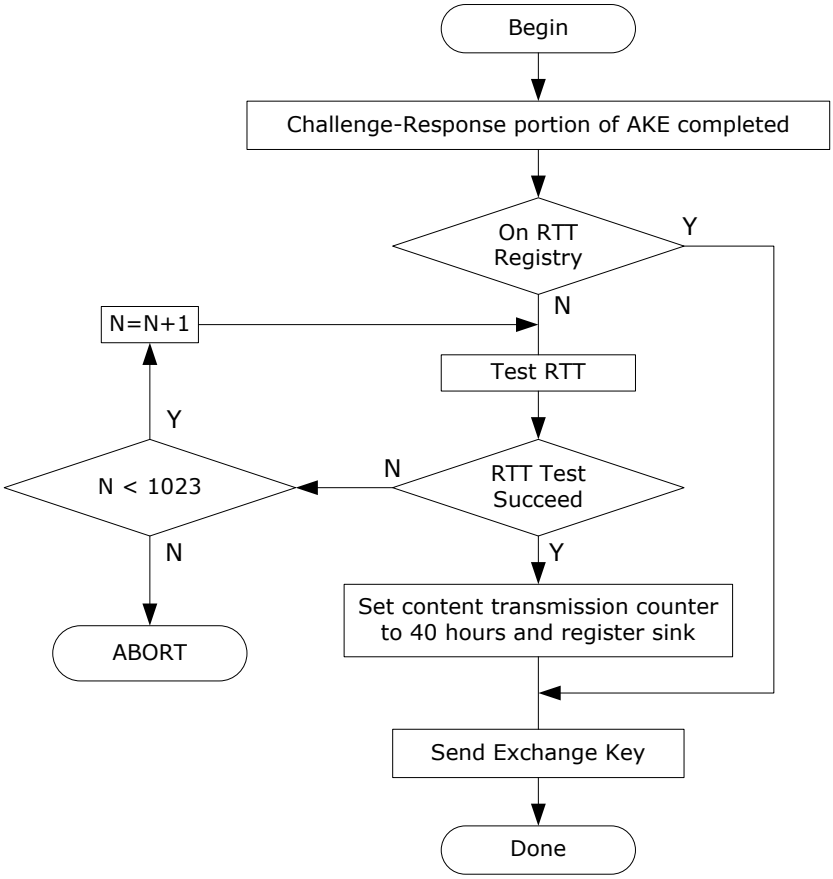
If RTT procedure is aborted, the source device shall not provide any Exchange Key.

4.6.5 RTT-AKE

The RTT-AKE procedure defined below shall be executed to share either a Session Exchange Key (K_S) or a Multicast Exchange Key (K_M) specified in Section 5.

The sink device after completing Challenge-Response portion of AKE will wait a next command from the source device and the sink device will abort if it receives any other command than the RTT_READY command, EXCHANGE_KEY command, or AKE_CANCEL command.

After completing Challenge-Response portion of AKE, the source device examines the RTT registry and if the sink device's Device ID is on its RTT registry, the source device proceeds to "Send Exchange Key" portion of AKE in the following figure otherwise the source device initiates a RTT test procedure and if during test it obtains a RTT measurement of 7 milliseconds or less it will add the sink device's Device ID to its RTT registry, set content transmission counter to 40 hours or less, and then proceed to "Send Exchange Key" portion of AKE. The source device may perform RTT test procedure to reset content transmission counter to initial value (40 hours or less) by succeeding the RTT test while the sink device's Device ID is on its RTT registry. If the CK flag is set to 1 in the sink device's certificate, ID_U shall be used instead of Device ID in above process.



5 Key and Content Management

5.1 Key and Crypto-variable Management

The following are the length of keys and constants described below:

Key or Constant	Size (bits)
Multicast Exchange Key (K_M) for 2 or more sinks	128
Session Exchange Key (K_S) unique per sink	128
Move Exchange Key (K_{XM})	128
Remote Exchange Key (K_R)	128
Scrambled Exchange Key (K_{SX})	128
Content Key (K_C)	128
Seed for Content Channel (N_C)	64

5.1.1 Exchange Keys

DTCP2 has several Exchange Keys that are used as input to calculating the Content Key:

- Multicast Exchange Key (K_M) is used to support simultaneous transports of same content to two or more sink devices (multicast).
- Session Exchange Key (K_S) is used to support transport of content to a single specific sink device.
- Move Exchange Key (K_{XM}) is used to support Move protocol.
- Remote Exchange Key (K_R) is used to support transport of content to remote devices.

Note that K_M, K_S, K_{XM}, and K_R are collectively described as the "Exchange Key", and K_S, K_{XM} and K_R are collectively described as the "Unicast Exchange Key".

Source devices shall separately generate the values for the Exchange Key.

The detail of the **exchange_key_label** (K_{XM}_label in Move Protocol) is described in the DTCP2 Specification available under license from the DTLA.

Source devices shall not send a Unicast Exchange Key to more than one sink devices. For this purpose, if a source device sends a Unicast Exchange Key to the same sink device more than once, the source device shall manage each Unicast Exchange Key with an identifier of a sink device to which such Unicast Exchange Key is provided, and ensure that a Unicast Exchange Key never be provided to more than a single sink device, where the identifier of sink device shall be Device ID for Unique-key Devices or Device ID with ID_U for Common-key Devices.

Source devices shall expire Multicast Exchange Key (K_M), Session Exchange Key (K_S), and Remote Exchange Key (K_R) within 2 hours after the end of the last content transmission from the source device.

Sink devices shall expire Multicast Exchange Key (K_M), Session Exchange Key (K_S), and Remote Exchange Key (K_R) within 2 hours of continuous non-use of any of these Exchange Keys for decryption unless content transmission associated with such Exchange Key is in progress.

Source and sink devices shall expire all Exchange Keys when they detect themselves being disconnected from all interfaces. For wireless interfaces this means when device detects that it is not connected to an access point or it is not directly connected to another device.

Expiration of K_R by source device based on the above rules shall be done even while a keep-alive timer for the K_R is counting.

Source devices shall not change or expire an Exchange Key (K_M, K_S, K_{XM}, or K_R) during content transmission using PCP(s) if that Exchange Key is assigned for that content transmission.

5.1.2 Exchange Key Transport

After the completion of Authentication[2], the source device establishes one of the Exchange Key other than Move Exchange Key (K_{XM}) described in Section 5.1.1. The following procedure shall be used to share an Exchange Key between a source and a sink device:

1. Source device uses RNG to generate an Exchange Key.
2. The source device then scrambles the key according to the function using K_{AUTH} described in the DTCP2 Specification available under license from the DTLA.
3. The source device sends the Scrambled Exchange Key (K_{SX}) to the sink device.
4. The sink device descrambles the K_{SX} according to the function using K'_{AUTH} described in the DTCP2 Specification available under license from the DTLA.

Note that the procedure to share Move Exchange Key (K_{XM}) is different from the above and specified in Section 6.1.2.

5.1.3 Content Encryption Algorithm

For content encryption, AES-128 using the Cipher Block Chaining (CBC) mode is used. AES-128 is described in FIPS 197 dated November 26, 2001 and the CBC mode is described in NIST SP800-38A 2001 Edition.

5.1.4 Content Key and Initialization Vector Calculation

The Content Key (K_C) and Initialization Vector (IV) for CBC mode shall be calculated using HMAC-SHA256 based on the NIST SP800-108 as specified in DTCP2 Specification available under license from DTLA.

Source devices shall reset CBC at the beginning of each PCP and sequentially apply IV to encryption of content in PCP.

The mechanism for establishing the Content Keys (K_C) used to encrypt/decrypt content being sent over DTCP2 is described in the DTCP2 Specification available under license from the DTLA, which includes definition.

[2] Authentication Procedure of Remote Access (RA-AKE) is specified in 6.2.2.

5.2 Content Usage Indicators

The following content usage indicators are used in the Usage Rules of the CMI packets (see Section 5.3.3.1). The definition and rules of these indicators is specified in EXHIBIT "B" of the "DTCP2 DIGITAL TRANSMISSION PROTECTION LICENSE AGREEMENT".

5.2.1 Copy Control Information (CCI)

This field indicates the CCI.

CCI	Meaning
00_2	Copy-Freely
01_2	No-More-Copies
10_2	Copy-One-Generation
11_2	Copy-Never

5.2.2 Encryption Plus Non-assertion (EPN)

This field indicates the value of the EPN.

- Only when CCI is Copy-Freely (00_2) is the EPN field valid. In other conditions, the EPN field is invalid.
- With the setting of the EPN field, Copy-Freely state has two variations; Copy-Freely with EPN-asserted (CF/EPN) and Copy-Freely with EPN-unasserted (CF).

EPN	Meaning
0_2	EPN-asserted
1_2	EPN-unasserted

5.2.3 Retention_State[3]

This field indicates the value of the Retention_State.

Retention_State	Retention Time
000_2	Forever
001_2	1 week
010_2	2 days
011_2	1 day
100_2	12 hours
101_2	6 hours
110_2	3 hours
111_2	90 minutes

5.2.4 Retention_mode

This field is used to indicate the mode of the Retention function in combination with the CCI as shown in following tables.

Modes	Retention_mode	CCI
Retention-mode	0_2	11_2
Non-Retention-mode	Other combinations	

[3] If an inter-industry standard or consensus supports retention states that differ from those set forth in this Specification, then this Specification may be amended or supplemented to reflect such consensus retention states.

5.2.5 Analog Protection System (APS)[4]

This field indicates the APS.

APS	Meaning
00_2	Copy-Freely (APS off)
01_2	APS is on : Type 1 (AGC)
10_2	APS is on : Type 2 (AGC + 2L Colorstripe[5])
11_2	APS is on : Type 3 (AGC + 4L Colorstripe[5])

5.2.6 Image_Constraint_Token (ICT)

This field indicates the value of the ICT.

ICT	Meaning
0_2	High Definition Analog Output in the form of Constrained Image
1_2	High Definition Analog Output in High Definition Analog Form

5.2.7 Analog_Sunset_Token (AST)

This field indicates the value of the AST.

AST	Meaning
0_2	AST-asserted
1_2	AST-unasserted

5.2.8 Digital_Only_Token (DOT)

This field indicates the value of the DOT.

DOT	Meaning
0_2	DOT-asserted
1_2	DOT-unasserted

5.2.9 Audio Enhancement Token (AET)

This field indicates the value of the AET.

AET	Meaning
0_2	AET-asserted
1_2	AET-unasserted

5.2.10 Copy Count (CC)

This field indicates the value of CC.

- Only when K_C and IV calculated with Session Exchange Key (K_S) or Move Exchange Key (K_{XM}) is used, CCI is No-More-Copies (01_2) and CC field has a non-zero value, the CC field is valid. In other conditions, the CC field is invalid.

CC	Meaning
0000_2	Invalid
Others	Copies up to the value of this field are allowed

[4] as described in the Specification of the Macrovision Copy Protection Process for DVD Products, Revision 7.1.D1, September 30, 1999

[5] 2L/4L Colorstripe is applicable on for NTSC analog output.

CC field rules for source devices:

- When source devices do not use the CC field, they shall set CC field to zero.

CC field rules for sink devices:

- When sink devices receive a content stream with valid CC field, they may record the content as the CC Content.
- When sink devices receive a content stream with invalid CC field, they shall ignore CC field.

Sink devices may make a request about the setting of the CC field in the following transmission to source devices with the CC_req value shown in the following table. Source devices that handle CC Content shall support the CC_req value of 0_{16} and should support the CC_req value of F_{16}. The recommended HTTP header field to send the CC_req value is specified in Section 10.3.6.

CC_req	Meaning
0_{16}	Non Copy Count encoding with the CC field of zero (i.e. No-More-Copies).
1_{16} – E_{16}	Copy Count encoding with the CC field having the value of the CC_req or smaller but maximum possible value.
F_{16}	Copy Count encoding with the CC field of the maximum possible value.

5.2.11 Standard_Digital_Output_Token (SDO)

This field indicates the value of the SDO. Note that the value of one means asserted.

SDO	Meaning
0_2	SDO-unasserted
1_2	SDO-asserted (Standard Digital Output in L2 protection is permitted)

5.2.12 High_Dynamic_Range_Token (HDR)

This field indicates the value of the HDR. Note that the value of one means asserted.

HDR	Meaning
0_2	HDR-unasserted (Standard Dynamic Range conversion is permitted)
1_2	HDR-asserted (Standard Dynamic Range conversion is not permitted)

5.2.13 L2_protection-Only_Token (L2-Only)

This field indicates the value of the L2-Only[6]. Note that the value of one means asserted. As specified in the DTCP2 Compliance Rules, devices that apply L1 protection[7] shall not handle content with L2-Only=1. For example, sink function with L1 protection shall not decrypt content if L2-Only=1.

L2-Only	Meaning
0_2	L2-Only-unasserted
1_2	L2-Only-asserted (L2 protection only)

[6] Details of L2 protection are described in EXHIBIT "C" ROBUSTNESS RULES – GENERAL of the DTCP2 Adopter Agreement.

[7] Details of L1 protection are described in EXHIBIT "C" ROBUSTNESS RULES – GENERAL of the DTCP2 Adopter Agreement.

5.2.14 Enhanced_Image_Token (EI)

This field indicates the value of the EI. Note that the value of one means asserted. As specified in the DTCP2 Compliance Rules, devices that apply L1 protection[7] shall not handle content with EI=1. For example, sink function with L1 protection shall not decrypt content if EI=1.

EI	Meaning
0_2	EI-unasserted (Non-Enhanced Image)
1_2	EI-asserted (Enhanced Image)

If video quality of a single content can be changed during content transmission, such as by adaptive bit rate control, this can cause alteration of EI_Token value, EI_Token for such content should be fixed to EI-asserted.

5.2.15 Retention Time

The value for this field represents time in minutes that the content may be retained from when the CMI packet including this field is received while such content shall be retained as a Bound Copy[8]. This field can have values from 0000_{16} to $FFFF_{16}$. Note the value 0000_{16} means no retention, and $FFFF_{16}$ means Forever. This field is used with Retention_mode field set to 1_2 (Non-Retention-mode) in CMI Descriptor 1 by overriding such setting.

Source devices that set non-zero value to Retention_Time field shall set CCI field to 11_2 (Copy-Never) and set Retention_mode to 1_2 (Non-Retention-mode) in CMI-Descriptor 1 contained in the same CMI Packet.

Sink devices shall not retain content based on this field if CCI field is not set to 11_2 (Copy-Never) or Retention_mode is not set to 1_2.

When Source device sends retained content received with Retention_Time field of $FFFF_{16}$ and FBCP field set to 1_2, the same setting of CMI Descriptor 2 shall be used.

When Source device sends retained content received with Retention_Time field with a non-zero value other than $FFFF_{16}$ and FBCP field set to 1_2, either of the following CMI Packet shall be used:

- CMI packet including CMI Descriptor 1 with CCI field set to 11_2 (Copy-Never) and Retention_mode set to 1_2 (Non-Retention-mode) without CMI Descriptor 2.
- CMI packet including CMI Descriptor 1 and CMI Descriptor 2 with the value of Retention_Time reduced by the retained time in the Source device and FBCP field set to 1_2.

When Source device sends retained content from which making further copy is not permitted, such as the one retained based on Retention_State field in CMI Descriptor 1, CMI Packet for that content shall not include Retention_Time field unless it is set to 0000_{16}.

5.2.16 Further Bound Copy Permitted (FBCP)

This field indicates whether a further Bound Copy can be made from a Bound Copy retained in accordance with the Retention_Time field. Note that this field when set to 1_2 indicates permission of making a further Bound Copy if the retention period has not expired and does not indicate permission of copying content made irrespective of the Retention_Time field.

When Source device sends retained content received with FBCP set to 0_2, the Source device shall send a CMI packet including CMI Descriptor 1 with CCI Field set to 11_2 (Copy-Never) and Retention_mode set to 1_2 (Non-Retention-Mode) without CMI Descriptor 2 for that content.

FBCP	Meaning
0_2	Further Bound Copy Prohibited
1_2	Further Bound Copy Permitted

[8] See the definition in 1.10.

5.2.17 Recording Duration

This field indicates, in unit of minutes, the duration of content associated with a CMI Descriptor 2. Sink devices can determine if rendering of received content is possible without termination using the values of both the Recording Duration field and the Retention Time field. If the duration of content can be extended, Source device sets the maximum value to the Recording_Duration field. Sink devices may change the value of the Recording_Duration field using the value of the actual duration of the recorded content.

5.3 Content Management Information

5.3.1 General

5.3.1.1 Purpose and Scope

Content Management Information (CMI) refers to usage rules associated with the content (e.g. CCI, DOT, Copy-count, etc.) which can be transmitted over DTCP2 as defined by DTLA.

The CMI is sent out in the CMI Descriptor within the CMI Field of CMI packet where each CMI Descriptor has its own ID, format and rules as defined by DTLA. Each unique CMI Descriptor ID refers to specific set of CMI. CMI Field consists of one or more CMI Descriptors. Source devices and sink devices shall follow the corresponding rules for each CMI Descriptor defined in the following sections. Source devices shall not send any CMI Descriptor which is not supported by themselves except the case of the DTCP2 Bridging and the case when DTLA approves.

Note that new CMI Descriptor may be additionally defined and transmitted with the CMI Descriptors currently defined. CMI descriptors allow for future expandability by either defining new descriptors or expansion of a currently defined descriptor. In the event that additional usage rule is added to a currently defined descriptor, the length of the CMI Descriptor Data field may be extended and the descriptor byte length value may be changed. If this occurs, currently defined fields in the CMI Descriptor Data shall not be altered to ensure backward compatibility. All devices shall be designed so that any change to the descriptor byte length value that results from an extension of the CMI Descriptor Data field shall not prevent access to contents of the CMI Descriptor Data field defined as of the time the device is manufactured.

Example 1. Suppose CMI Descriptor-X has a set of usage rules. CMI Descriptor-Y is defined with a set of other usage rules later. A source device which supports the CMI Descriptor-X and CMI Descriptor-Y sends both CMI Descriptor-X and CMI Descriptor-Y. When a sink device does not support CMI Descriptor-Y, the sink device may use the received content in accordance with the usage rules defined in the CMI Descriptor-X if the sink device supports the CMI Descriptor-X.

Example 2. Suppose CMI Descriptor-A has a set of usage rules, CMI Descriptor-B has another set of usage rules, and CMI Descriptor-C is defined with a set of other rules later. A source device which supports all of these CMI Descriptors sends CMI Descriptor-A and CMI Descriptor-C in a CMI packet. A sink device that supports both CMI Descriptor-A and CMI Descriptor-C may use the received content from that source device by referring to the usage rules in the CMI Descriptor-C, and ignoring the CMI Descriptor-A.

Example 3. Suppose CMI Descriptor-D has a set of usage rules D-1, D-2 and D-3. CMI Descriptor-E with a set of usage rules D-1, D-2, D-3 and D-4 is defined later where the usage rule D-4 is defined to be ignorable by a sink device. A new source device which supports both the old CMI Descriptor-D and the new CMI Descriptor-E can send both CMI Descriptor-D and CMI Descriptor-E. An old sink device which only supports CMI Descriptor-D can use the received content in accordance with the old CMI Descriptor-D.

5.3.1.2 General Rules for Source Devices

The following are the rules for source devices:

- Source devices shall use the Content Key for the encryption of content associated with the latest CMI Packet sent to sink device(s).
- When source devices send more than one CMI Descriptors in a CMI packet, the following rules shall be kept:
    - CMI Descriptors shall be sent in ascending order of CMI Descriptor ID contained in each CMI Descriptor.
    - The same CMI Descriptor shall not be contained in a single CMI packet. CMI Descriptors shall be concatenated without any space.
    - The set of CMI Descriptor IDs in a CMI packet shall not be changed although the value of CMI Descriptor Data may be changed during content transmission.
- When source devices send content including multiple streams CMI Descriptors which provide a single set of content management information for all streams, such as the CMI Descriptor 1, shall not be used unless information in the CMI Descriptor is consistent with the content management information of any streams.
- Source devices shall send a CMI packet before transmitting DTCP2 protected content using PCP. Following the CMI packet, source devices may send one or more sequential separate PCPs without a CMI packet as long as the content management information in the CMI packet is applicable to the PCPs. Source devices may update data of the CMI Descriptor in the next CMI packet in the middle of content transmission using PCP. Source devices may send the same CMI packet as the previous one even if the upstream does not change the usage rules.
- For RTP transfers, each RTP payload is encapsulated by a single PCP and a single CMI packet is transmitted in an RTP packet. When new CMI Descriptor is defined in the future and the size of a CMI packet is larger than the maximum size of RTP payload, the CMI Field could be split into several RTP payloads.
- For HTTP transfers, source devices shall send CMI packets and PCPs (associated to the CMI packets) in the same TCP connection.

5.3.1.3 General Rules for Sink Devices

The following are the rules for sink devices:

- When sink devices receive content associated to a CMI packet, they shall handle the content in accordance with the content management information contained in the CMI packet. Unless specified by a CMI Descriptor rule, sink devices shall use only the information in CMI packet as usage rules.
- When sink devices receive a CMI packet that contains more than one CMI Descriptor,
    - Sink devices shall use the usage rules of only one of the supported CMI Descriptors and ignore the other CMI Descriptors. Sink devices may select any one of the supported CMI Descriptors.
- For Bridge Devices, even if they support none of the received CMI Descriptor(s), they may output the content with the same CMI packet.
- When a sink device does not support any of the CMI Descriptors in a CMI packet, it shall discard all content associated to the CMI packet.
- Sink devices shall apply the usage rules indicated by a CMI packet to the following PCP(s) to which the CMI packet is associated until the next CMI packet.
- Sink devices that only have sink function(s) with the L1 protection[9] shall not decrypt PCPs if CMI Descriptor(s) associated to such PCPs contains the L2-Only field of one (asserted) or the EI field of one (asserted).

[9] Details of L1 protection are described in EXHIBIT "C" ROBUSTNESS RULES – GENERAL of the DTCP2 Adopter Agreement.

5.3.2 CMI Field

CMI Field may consist of one or more CMI Descriptors. Every CMI Descriptor shall be one byte aligned. CMI Descriptors shall be contained in ascending order of CMI Descriptor ID.

An example of CMI Field is described in the following figure.

	msb							lsb
CMI Field[0..M-1]	CMI Descriptor X							
CMI Field [M..M+N-1]	CMI Descriptor Y							

CMI Field [0..M-1]: Contains CMI Descriptor X format data.

CMI Field [M..M+N-1]: Contains CMI Descriptor Y format data.

5.3.3 CMI Descriptor Descriptions

5.3.3.1 CMI Descriptor General Format

The general format of CMI Descriptor is as follows:

	msb							lsb
CMI Descriptor ID [0]	ID							
Extension [0]	Extension							
Byte Length [0]	Byte length of CMI Descriptor Data (16 bits)							
Byte Length [1]								
CMI Descriptor Data [0]	Usage Rules							
-								
CMI Descriptor Data [N-1]								

CMI Descriptor ID [0]: Contains ID value of CMI Descriptor assigned by DTLA.

Extension [0]: Is specified by each CMI Descriptor.

Byte Length [0..1]: Denotes byte length of Usage Rules field, where it is less than or equal to 64KB.

CMI Descriptor Data [0..N-1]: Represents usage rules and there is no usage rule when Byte Length field is zero.

When sink devices receive CMI Descriptor which has the Extension field of non-zero value, sink devices shall regard it as unsupported CMI Descriptor, and shall ignore the following fields. Note that this CMI Descriptor may have extended Byte Length field by using Extension field that may be more than 64KB.

5.3.3.2 CMI Descriptor 0

The format of CMI Descriptor 0 is as follows:

	msb							lsb
CMI Descriptor ID [0]	ID (00_{16})							
Extension [0]	00_{16}							
Byte Length [0]	Byte length of CMI Descriptor 0 data (0001_{16})							
Byte Length [1]								
CMI Descriptor Data [0]	Reserved (0000_2)				C_T			

Reserved is a field for future extension where source devices shall set the value zero for every bit of Reserved field unless otherwise noted. Sink devices shall use value of Reserved field to calculate K_C in order that they can accommodate any future changes.

C_T field indicates the type of content that is associated to this CMI Descriptor 0 and has the following values:

C_T	Meaning
0000_2	Audiovisual content

0001_2	Audio content
$0010_2..1111_2$	Reserved

Rules for source devices:

- Source devices that handle Audio content shall support and always insert this CMI Descriptor to CMI packets when the source devices transmit Audio content. Source devices that only support Audiovisual content do not have to support this CMI Descriptor.

Rules for sink devices:

- When sink devices use this CMI Descriptor, sink devices shall not handle content in a way that needs to refer to any usage rules associated to that content, such as recording based on the CCI and analog video output.
- Sink devices with rendering functions of Audio content shall support this CMI Descriptor.
- Sink devices shall regard this CMI Descriptor as unsupported when the sink device does not support the value in the C_T field.

5.3.3.3 CMI Descriptor 1

Source devices shall use this CMI Descriptor in transmissions of audiovisual content, and sink device shall support this CMI Descriptor as the baseline CMI Descriptor unless another CMI Descriptor is required as the mandatory CMI Descriptor. CMI Descriptor 1 is used **only with audiovisual content**. The format of CMI Descriptor 1 is as follows:

	msb							lsb
CMI Descriptor ID [0]	ID (01_{16})							
Extension [0]	00_{16}							
Byte Length [0]	Byte length of CMI Descriptor 1 data (0003_{16})							
Byte Length [1]								
CMI Descriptor Data [0]	RES(1_2)	RM	Retention_State			EPN	CCI	
CMI Descriptor Data [1]	AET	RES(11_2)			DOT	AST	ICT	APS
CMI Descriptor Data [2]	SDO	HDR	L2-Only	EI	CC			

RES are fields for future extension where source devices shall set the value to one for every bit of the RES field unless otherwise noted. Sink devices shall use value of RES fields to calculate K_C and IV in order that they can accommodate any future changes.

RM field indicates Retention_mode as described in Section 5.2.

Retention_State field indicates Retention_State as described in Section 5.2.

EPN field indicates Encryption Plus Non-assertion as described in Section 5.2.

CCI field indicates CCI as described in Section 5.2.

AET field indicates Audio Enhancement Token as described in Section 5.2.

DOT field indicates Digital Only Token as described in Section 5.2.

AST field indicates Analog_Sunset_Token as described in Section 5.2.

ICT field indicates Image_Constraint_Token as described in Section 5.2

APS field indicates analog copy protection information as described in Section 5.2.

SDO field indicates SDO Token as described in Section 5.2.

HDR field indicates HDR Token as described in Section 5.2.

L2-Only field indicates L2-Only Token as described in Section 5.2.

EI field indicates EI Token as described in Section 5.2.

CC field indicates Copy-count as described in Section 5.2.

5.3.3.4 CMI Descriptor 2

The format for CMI-2 is as follows:

	msb							lsb
CMI Descriptor ID [0]	ID (02_{16})							
Extension [0]	00_{16}							
Byte Length [0]	Byte length of CMI Descriptor 2 data (0004_{16})							
Byte Length [1]								
CMI Descriptor Data [0]	Retention_Time							
CMI Descriptor Data [1]								
CMI Descriptor Data [2]	FBCP	Reserved (00_2)		Recording _Duration (13 bits)				
CMI Descriptor Data [3]								

Retention_Time field indicates Retention Time as described in Section 5.2.15.

FBCP field indicates the Further Bound Copy Permitted described in Section 5.2.16.

Recording_Duration field indicates Recording Duration as described in Section 5.2.17.

When Source device sends CMI Descriptor 2, CMI Descriptor 2 shall follow CMI Descriptor 1 in a CMI Packet at the beginning of content. For example, if some TV programs are transmitted continuously, CMI packet for each program must be sent at the top of each program.

When Sink device uses a usage rule of CMI Descriptor 2, the Sink device shall handle the content in accordance with the usage rules in both CMI Descriptor 1 and CMI Descriptor 2. Note that ignoring CMI Descriptor 1 and only referring to CMI Descriptor 2 is not permitted.

5.4 CMI Packet and PCP Formats

5.4.1 Content Management Information Packet Format

The format of the Content Management Information (CMI) Packet is described in the following figure.

	msb							lsb
Header [0]	Packet_Type		Reserved (000000_2)					
Header [1]	CMI_Counter							
Header [2]	Reserved (zero)							
Header [3]	Reserved (zero)							
Header [4]	Byte length of the whole CMI Field (32 bits)							
Header [5]								
Header [6]								
Header [7]								
Body [0]	CMI Field							
Body [1]								
Body [2]								
-								
-								
-								
Body [X-1]								

Header [0]: Contains:

- **Packet_Type** field which has the fixed value of 01_2 for the CMI packet.
- **Reserved** field which has a value of zero.

Header [1]: Contains **CMI_Counter** field.

CMI_Counter is used to inform the sink device that data in the CMI packet has changed. If a source device sends the same CMI packet or detects that the values in both the Header and Body fields of the CMI packet has not changed since the most recent packet the source device sent, the source device shall not change the value of CMI_Counter, otherwise the source device shall increase the value of CMI_Counter of new CMI packet by one. When the value of CMI_Counter reaches 255, it shall wrap to zero for the next value. The initial value of the CMI_Counter should be set to a random value.

Header [2]: Contains **Reserved** field.

Header [3]: Contains **Reserved** field.

Header [4..7]: Denotes byte length of CMI Field.

Body [0..X-1]: Represents **CMI Field** as defined in Section 5.3.2.

5.4.2 Protected Content Packet Format

DTCP2 encrypted content is sent via Protected Content Packet (PCP). The general format of PCP is described below:

	msb							lsb
Header [0]	Packet_Type		Format		Reserved (000_2)			ENC
-	Format Dependent Data							
-								
-								

Header [0]: contains:

- **Packet_Type** field set 11_2 for DTCP2 PCP packet.
- **Format** field indicates data format after Header [0] as defined in the following table. The source device shall not change the value of the format field during content transmission.

Format	Meaning
00_2	Baseline PCP
$01_2..11_2$	Reserved

- **Reserved** field is set to zero.
- **ENC** bit indicate whether or not the content is encrypted, where ENC = 1 means encrypted. ENC bit shall be set to one when the CCI value is not zero or EPN value is zero in the CMI packet associated with this PCP. Otherwise ENC bit shall be set to zero.

5.4.2.1 Baseline PCP Format

The format for a Baseline PCP (Format = 00_2) is as follows:

	msb							lsb
Header [0]	11_2		00_2		Reserved (000_2)			ENC
Header [1]	exchange_key_label							
Header [2]	CMI_Counter							
Header [3]	N_C (64 bits)							
-								
Header [10]								
Header [11]	IV (128 bits)							
-								
Header [26]								
Header [27]	Byte length of content denoted as CL (32 bits)							
-								
Header [30]								
EC [0]	Content affixed with 0 to 15 bytes of padding							
EC [1]								
EC [2]								
-								
-								
-								
EC [N-1]								

Header [1]: Contains **exchange_key_label** field which is described in Section 5.1.1.

Header [2]: Contains **CMI_Counter** field which has the value of the CMI_Counter field in CMI Packet associated to this PCP.

Header [3..10]: Contains $\mathbf{N_C}$ field as described in Section 5.1.4.

Header [11..26]: Contains **IV** field as described in Section 5.1.4.

Header [27..30]: Denotes byte length of content (**CL**) and does not include any padding bytes, where CL shall be less than or equal to 512MB (20000000_{16}).

EC [0..N-1]: Represents encrypted frame[10] and there is no EC when CL is zero otherwise it is a multiple of 16 Bytes in length where N = (Int((CL-1)/16)+1)*16 where padding length is equal to N-CL and Int(X) means maximum integer less than or equal to X. The value of each padding Byte is 00_{16}.

[10] Cipher Block chaining resets every PCP. The IV described in section 5.1.4 .is used in an initial step in the encryption/decryption of every PCP.

6 Standard Functionality

6.1 Move Protocol

This section specifies a transaction based Move protocol for a Move function that uses a Move Exchange Key (K_{XM}) for each Move transaction. The transaction based Move protocol results in either the content being completely moved to the sink device (Success case) otherwise the content remain usable in the source device with no usable content in the sink device (Cancel case).

The Move protocol consists of three parts; Move RTT-AKE, Move Transmission and Move Commitment. Each transaction based Move protocol (Move transaction) begins with Move request from a sink device and completes when the Move Commitment process completes or any one of these processes are canceled or aborted.

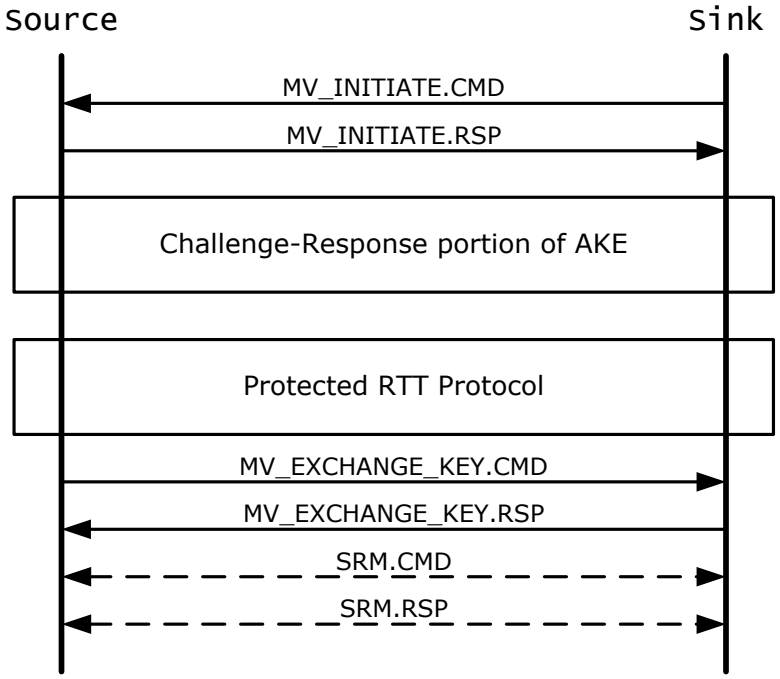
A unique Move Exchange Key (K_{XM}) is generated specifically for each Move transaction during Move RTT-AKE. K_{XM} is used to calculate the Content Key (K_C) used to encrypt the moved content. Content received by the sink device remains unusable until the successful completion of the Move Commitment phase of the Move transaction. Upon successful completion of the Move Commitment phase the moved content in the source device is made unusable and the moved content in the sink device is made usable.

Both source and sink devices can cancel a Move transaction any time before starting the Move Commitment process.

Bridge Devices that use a device certificate with AP flag of one shall neither request nor accept Move RTT-AKE.

6.1.1 Move RTT-AKE

Source devices generate K_{XM} specifically for the Move transaction and to calculate the Content Key (K_C) and IV used to encrypt the content to be moved during the Move transaction. The Move RTT-AKE is used to exchange K_{XM} and associated protocol flow is shown in following figure.



1. The sink device initiates the Move RTT-AKE protocol by sending MV_INITIATE command. If the source device can perform the Move protocol, the source device returns response as accepted.
2. Challenge-Response portion of AKE and Protected RTT protocol (see Section 4.6.4) are executed subsequently to share Authentication Key for Move (HK_{AUTH}). In the Challenge-Response portion of AKE, sink device sets only Bit 7 (K_{XM}) to one in the exchange_key field in the CHALLENGE command (even in the Remote Access) and source device performs the sink counting specified in Section 9. Source devices may skip Protected RTT Protocol when sink device is on its RTT Registry as specified in Section 4.6.1 or

if the Device ID or ID_U (if common key device) obtained during "Challenge-Response portion of AKE" exists in the RAC Registry for remote access.

3. The source device generates K_{XM} and sends it to the sink device. (See the following section for detail)

6.1.2 Establishing Move Exchange Key

The source device generates K_{XM} and sends it to sink device using the following procedure:

4-1. The source device shall assign a random value for the K_{XM} (using RNG) being established. The source device assigns K_{XM}_label to this K_{XM}.

4-2. The source device then scrambles the key (K_{XM}) as specified in DTCP2 Specification available under license from DTLA.

4-3. The source device sends K_{SXM} and K_{XM}_label to the sink device.

4-4. The sink device descrambles the K_{SXM} as specified in DTCP2 Specification available under license from DTLA.

Source devices use the value of K_{XM}_label to identify the corresponding Move transaction in the Move Transmission and Move Commitment processes. Source devices shall not reuse the value of K_{XM}_label assigned to the Move transaction(s) that have not yet completed.

Source and sink devices shall manage K_{XM} and K_{XM}_label as follows:

- K_{XM} shall be managed independent of the other Exchange Keys in terms of generation and expiration. K_{XM}_label may have the same value as the exchange_key_label of the other Exchange Keys.
- K_{XM} and K_{XM}_label can only be used in the corresponding Move transaction and shall not be used for other purposes.
- K_{XM} and K_{XM}_label shall be expired when the corresponding Move transaction completes regardless of result.
- It is mandatory that the source device expires a K_{XM} within 2 hours after Move Transmission using the K_{XM} has ceased.
- It is mandatory that the sink device expires a K_{XM} within 2 hours of continuous non-use of that K_{XM} for decryption.
- Source and sink devices must expire their K_{XM} when they detect themselves being disconnected from all interfaces. For wireless interfaces this means when device detects that it is not connected to an access point or it is not directly connected to another device.
- When K_{XM} is expired, the K_{XM}_label shall also be expired except when the K_{XM}_label is stored for resumption of Move Commitment (see Section 6.1.5).

The source device shall not reset the Sink Counter (see Section 9) when a Move Exchange Key is expired except for the case that the source device does not share any other Exchange Key (K_M, K_S, or K_R) after expiring the Move Exchange Key.

6.1.3 Move Transmission

The Move Transmission process starts upon the completion of the Move RTT-AKE and is part of the Move transaction where the moved content is encrypted using the K_C, calculated using K_{XM} in Move Transmission. The source device shall set the value of K_{XM}_label to the exchange_key_label field in each PCP.

Source devices shall not encrypt the same part[11] of content more than once using K_{XM} during a Move transaction. Source devices shall prevent content from plural transmission for move unless otherwise noted.

[11] The content may be retransmitted in transport protocol (ex. TCP).

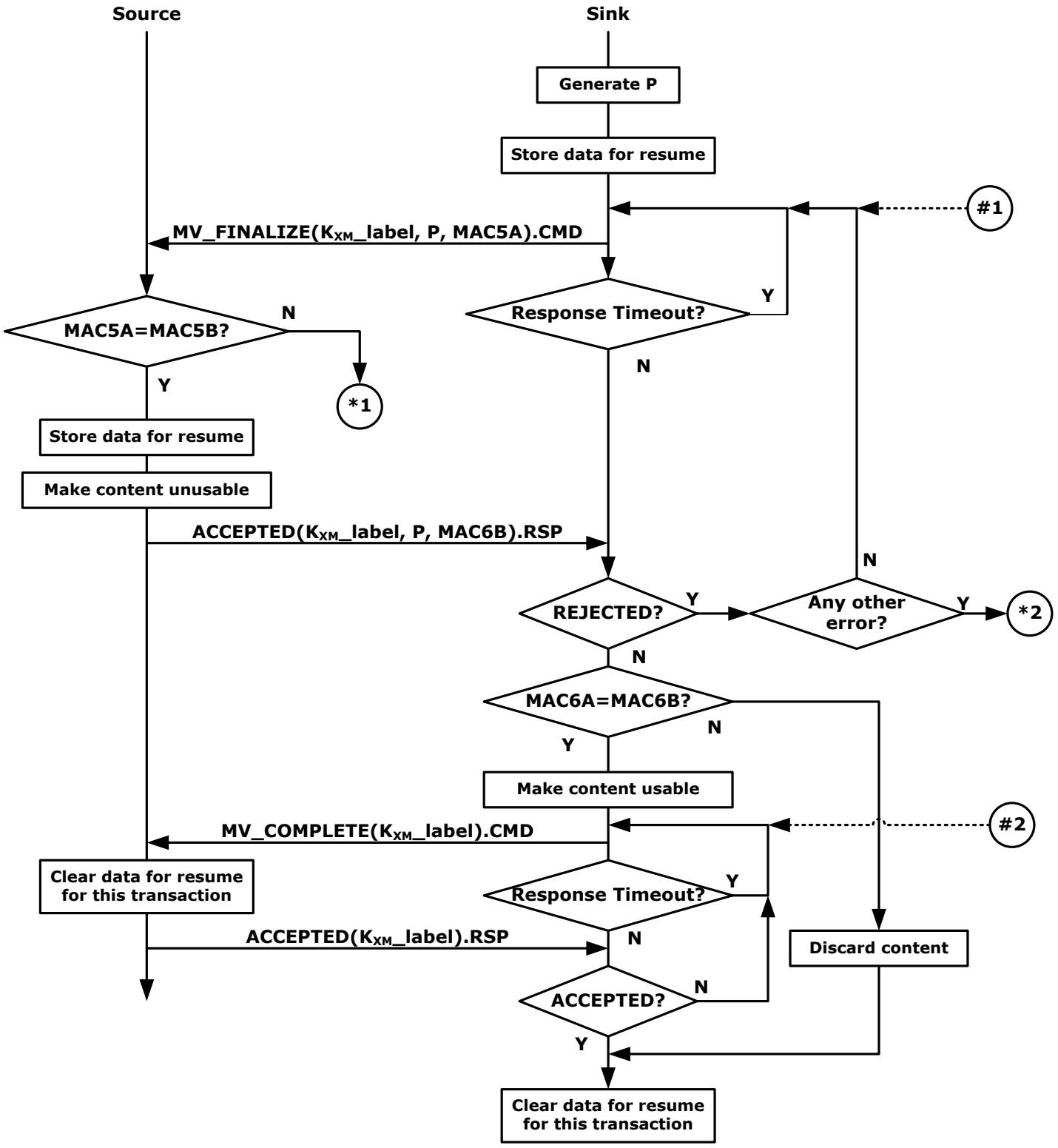
Sink devices shall keep the content received during Move Transmission unusable until successful completion of the Move Commitment process.

Refer to Section 10 for recommended HTTP header field.

6.1.4 Move Commitment

Sink devices initiate the Move Commitment process when Move Transmission has completed.

Sink device can make received content usable only upon the successful completion of the Move Commitment process. The following figure depicts the Move Commitment protocol flow.



***1** Source device is recommended to return REJECTED.RSP with "Any other error" status and keep waiting for MV_FINALIZE.CMD. However, it may cancel the Move transaction if the content has not yet been made unusable, then it should return REJECTED.RSP with "Any other error" and clear resume data for this transaction (if stored).

***2** Sink device is recommend to resend MV_FINALIZE.CMD after reconfirming IP address of source device with which K_{XM} has been exchanged. However, it aborts the Move Commitment process if result is the same. When it aborts, it should clear resume-data for this transaction.

SHA-256 is used to construct following MAC values that are exchanged during the Move Commitment protocol to ensure that the source device and the sink device share K_{XM}. The functions to calculate MAC values based on the K_{XM} are described in the DTCP2 Specification available under license from the DTLA.

P is a 64 bit random number (generated by RNG).

The source device computes MAC5B and compares it to MAC5A when MV_FINALIZE command is received. If not equal, the source device returns REJECTED response with "Any other error" status; else if equal, it shall make content transmitted in the Move Transmission unusable and returns ACCEPTED response to the sink device.

The sink device computes MAC6A and compares it to MAC6B when ACCEPTED response is received. If not equal, the sink device completes the Move transaction and discards any received content; else if equal, it makes content received in the Move Transmission usable and sends the MV_COMPLETE command to the source device.

When the sink device detects a timeout before receiving the ACCEPTED response to the MV_FINALIZE command, it should resend the MV_FINALIZE command unless REJECTED response with "Any other error" status is received from the source device with which K_{XM} was exchanged.

Source device completes the Move transaction after sending the ACCEPTED response when the MV_COMPLETE command is received. Sink device completes the Move transaction when the ACCEPTED response is received.

When a sink device detects a timeout before receiving the ACCEPTED response to the MV_COMPLETE command, it should resend the MV_COMPLETE command not to leave data for the Move Commitment process in sink device (and source device).

6.1.5 Resumption of Move Commitment

There is a brief period in the Move Commitment process where Moved content is marked unusable in both the source and sink devices such that if an interruption (e.g. loss of TCP connection) were to occur at this point in the process it would result in loss of moved content. To avoid this, it is recommended that both source and sink device store[12] required data[13] to complete Move Commitment protocol into non-volatile memory and perform the following resume procedure. The data is stored at the beginning and cleared at the end of the Move Commitment protocol as shown in Section 6.1.4.

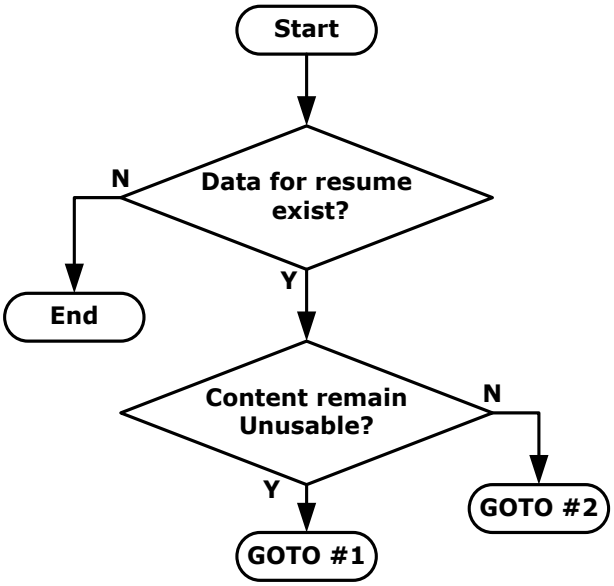
In case of a broken AKE TCP connection, the TCP connection must first be reestablished between the affected source and sink devices. When the sink device cannot get a DTCP2 Socket without notification from source device (e.g. content-push type Moves), the source device should transmit HTTP POST request[14] with DTCP2 Socket in the POST header to the sink device.

[12] At least the device should keep the stored data while the device is power-on.

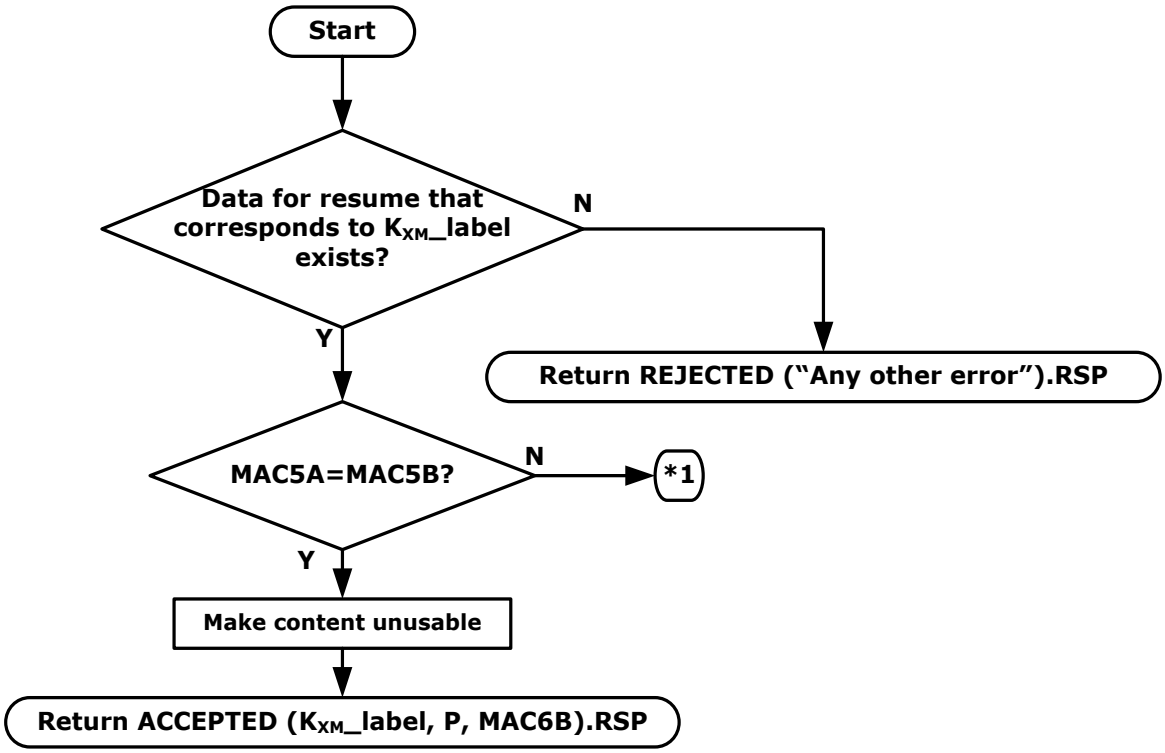
[13] For example, parameters required in Move Commitment and information to discover device and moved content. Note that to keep this information unchanged is essential for resume of Move Commitment (e.g. UPnP AV CDS Object ID).

[14] To the same destination as Move Transmission without message-body.

The sink device should execute the procedure depicted in the following (resume procedure for sink device) flow diagram after communication with the source device is reestablished and where #1 and #2 are the entry points specified in the figure in Section 6.1.4.

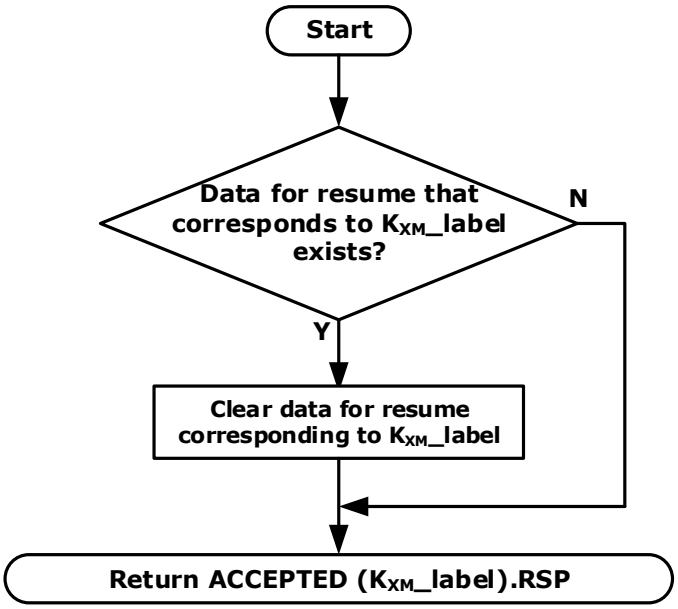


The following figure depicts the resume procedure for source device when MV_FINALIZE is received. The source device should execute the procedure based on the K_{XM}_label specified in the MV_FINALIZE command or the MV_COMPLETE command when one of these two commands is received.



***1** Source device is recommended to return REJECTED.RSP with "Any other error" status and keep waiting for MV_FINALIZE.CMD. However, it may cancel the Move transaction if the content has not yet been made unusable, then it should return REJECTED.RSP with "Any other error" and clear resume data for this transaction (if stored).

The following diagram depicts the resume procedure for the source device when MV_COMPLETE.CMD is received.



The source device should return the ACCEPTED response to the MV_COMPLETE command even when it has already cleared data for resume.

6.1.6 Cancel of Move Transaction

Source devices may cancel the Move transaction without disabling its content before issuing the first ACCEPTED response to the MV_FINALIZE command. Sink devices may cancel Move transaction as if it has received no content before issuing the first MV_FINALIZE command.

Sink devices which cancel a Move transaction shall discard content received during the Move Transmission in the Move transaction.

During the Move RTT-AKE process, the device desiring to cancel the Move transaction should send the AKE_CANCEL command.

During the Move Transmission process, the device desiring to cancel the Move transaction should send the MV_CANCEL command. It is recommended that source and sink devices maintain the AKE TCP connection until completion of Move Commitment or transmission of the MV_CANCEL command from source device.

During the Move Commitment process, source devices should return the REJECTED response with "Any other error" Status to the MV_FINALIZE command when it cancels the Move transaction. Source devices shall not return the REJECTED response with "Any other error" Status to the MV_FINALIZE command to cancel[15] the Move transaction if it has already issued the ACCEPTED response for the MV_FINALIZE command of the Move transaction. Source and sink devices shall clear data stored for resume corresponds to the Move transaction being canceled.

[15] Source devices that do not support the resumption of the Move Commitment may return the REJECTED response with "Any other error" status.

6.2 Remote Access

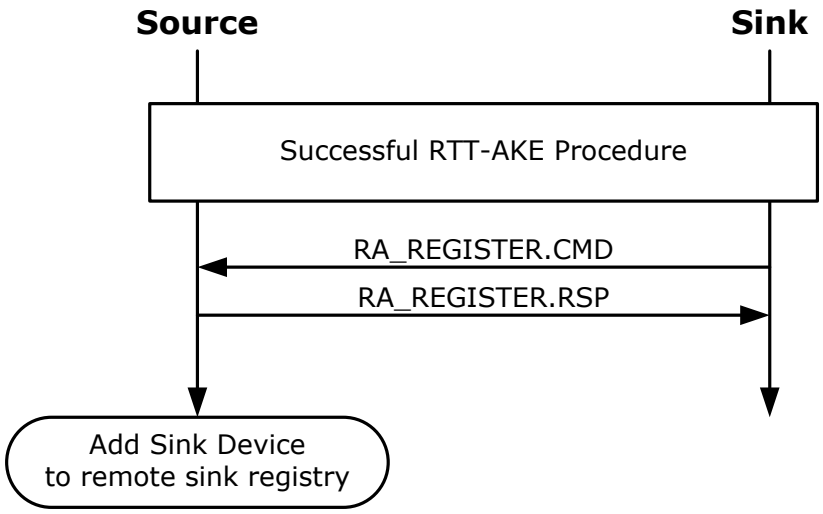
Remote access refers to the case where Remote Access capable source devices permit sink devices to request and connect to the source device without executing Localization procedures if the sink device is on the source device's Remote Sink Registry.

Remote Access capable source device requirements:

- Source devices must maintain a Remote Sink Registry.
- Source devices will add only those sink devices that successfully pass the registration protocol in Section 6.2.1 to the Remote Sink Registry by recording the Sink-ID which is either the Device ID of the sink device's Certificate or ID_U of the sink device.
    - For sink devices with common keying material, the source device shall record the ID_U instead of the Device ID of each sink device.
- The Remote Sink Registry is limited to 20 devices.
    - Record(s) in the Remote Sink Registry may be removed as needed.
- Source devices will permit only the prescribed number[16] (Maximum number of Remote Access Connection Counter: $RACC_{MAX}$) of remotely connected sink device(s) at any time as prescribed in the DTCP2 Adopter Agreement.
- Source devices will only permit AKE without RTT testing and TTL checking to proceed if the sink device's Sink-ID is contained in the source device's Remote Sink Registry.
- Bridge Devices must not transmit content via Remote Access simultaneous with its reception from an upstream source device.

6.2.1 Remote Sink Registration

Source devices will add a sink device to their Remote Sink Registry only if the connected sink device successfully passes the Remote Sink Registration protocol as described in the following figure:



The Remote Sink Registration procedure is as follows:

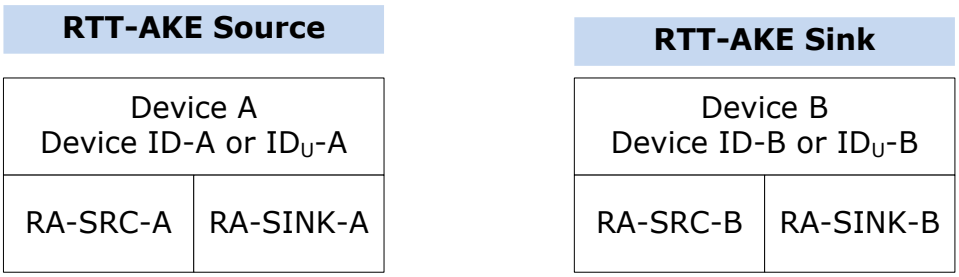
1. After completing RTT-AKE procedure successfully, sink device sends its Sink-ID (Device ID or ID_U) using RA_REGISTER.CMD.
2. Source device checks that the Sink-ID is the same as the Device ID if the CK flag is zero in the sink device's Device Certificate, or the Sink-ID is the same as the ID_U if such CK flag is one where such Device ID or ID_U is the one received in the RTT-AKE procedure completed immediately before.
3. Source device checks whether the Sink-ID has already been stored in its Remote Sink Registry. Skip the following steps 4, and 5 if the Sink-ID is already stored.
4. If the Sink-ID has not been registered, source device checks whether its Remote Sink Registry is not yet full.

[16] Refer to Section 2.6.1 of DTCP2 DIGITAL TRANSMISSION PROTECTION LICENSE AGREEMENT, Exhibit B, Part 2. $RACC_{MAX}$ is either 1 or a number permitted by an indication, such as a flag or descriptor.

5. If checks in both step 2 and 4 are passed, source device adds the Sink-ID to its Remote Sink Registry.
6. Source device returns RA_REGISTER.RSP with the result of registration.

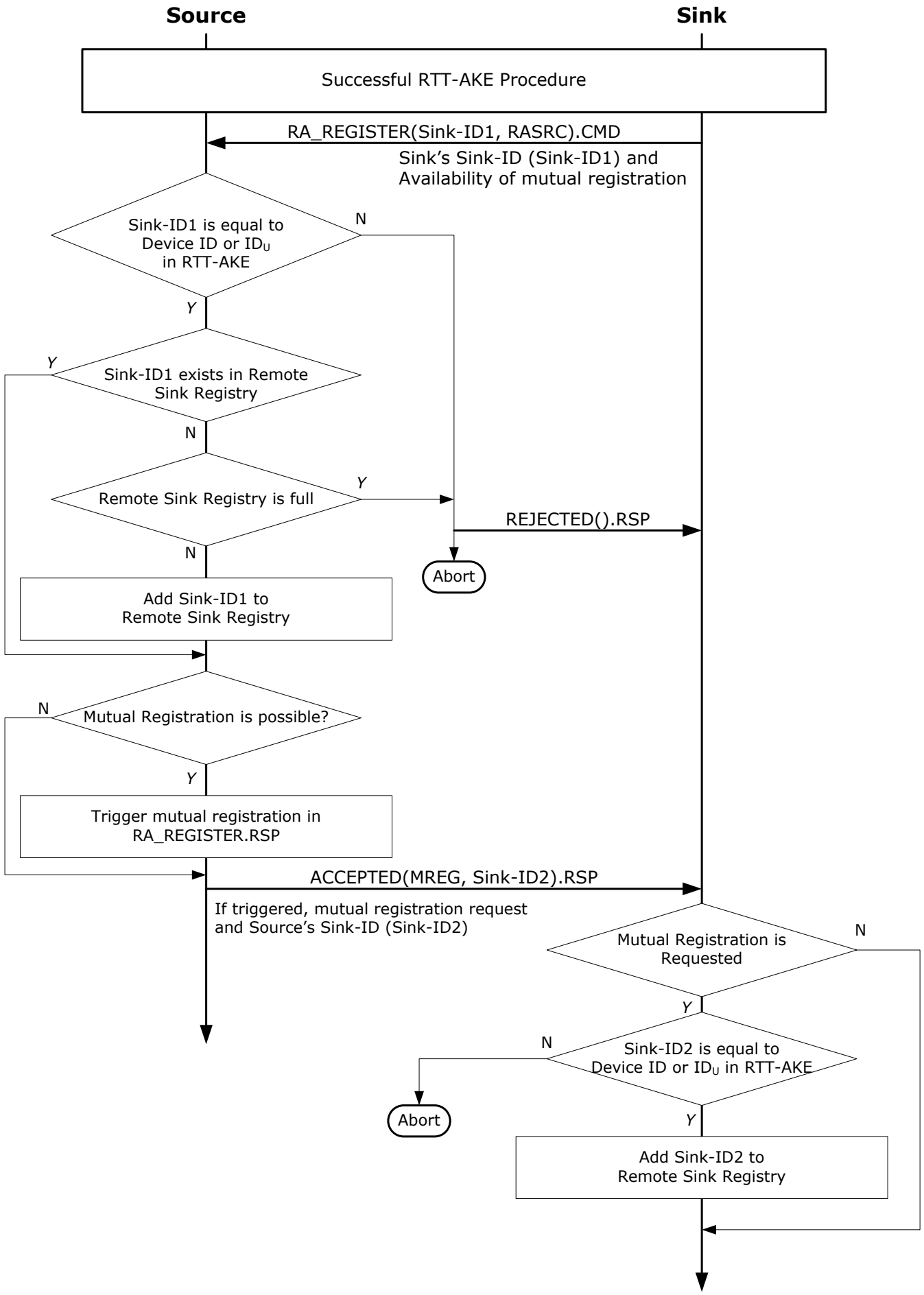
6.2.1.1 Mutual Registration

Between devices that have capabilities of both Source Function (RA-SRC) and Sink Function (RA-SINK) capability of remote access, mutual registration is possible in a single Remote Sink Registration procedure as long as the Source Functions in both devices can add another Sink-ID to their Remote Sink Registries. For avoidance of doubt, source devices may not always request mutual registration when it is possible.



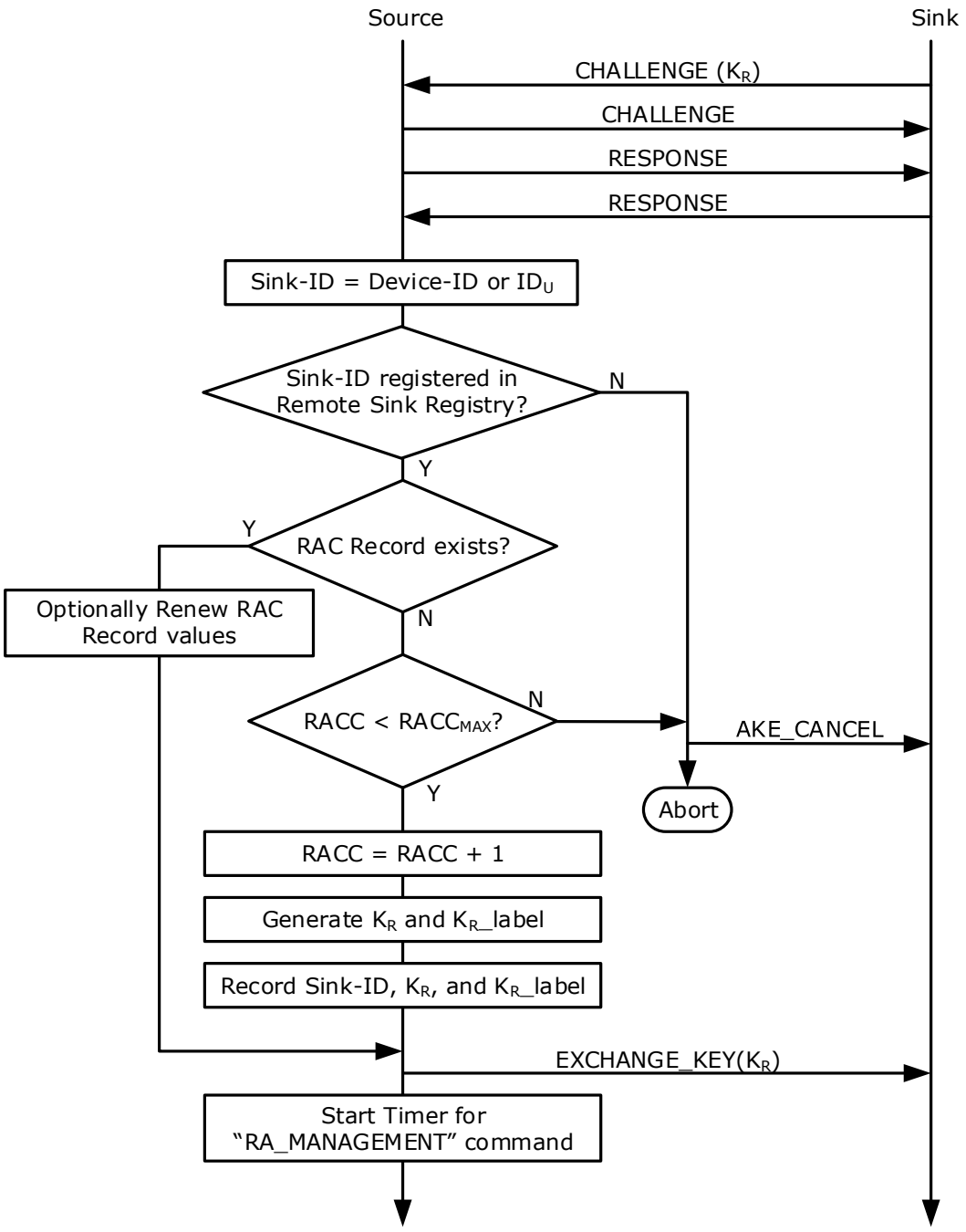
In mutual registration, device A's RA-SRC-A registers device B's Sink-ID and the device B's RA-SRC-B registers device A's Sink-ID as Remote access sink in parallel. It is executed when device B's RA-SRC-B declares that the mutual registration is acceptable in RA_REGISTER.CMD and device A's RA-SINK-A requests the mutual registration in RA_REGISTER.RSP that also includes device A's Sink-ID. Note that device B shall not declare that mutual registration is acceptable to multiple devices in parallel, and device A with common device certificate shall set its ID_U in the X_{IDU} field of the RESPONSE command in RTT-AKE procedure. In the case of mutual registration, the following additional steps are continued after step 5 in the above Remote Sink Registration procedure:

6. Source device returns REJECTED response and aborts the mutual registration if the check in step 2 or 4 fails, otherwise it returns ACCEPTED response with the source device's Sink-ID (Device ID) and flag to request mutual registration when sink device declares that mutual registration is acceptable.
7. Sink device checks that the source device's Sink-ID is the same as the Device ID if the CK flag is zero in the source device's Device Certificate, or the Sink-ID is the same as the ID_U if such CK flag is one where such Device ID or ID_U is the one received in the RTT-AKE procedure that immediately preceded remote registration procedure.
8. Sink device checks whether or not the source device's Sink-ID has already been stored in its Remote Sink Registry, and adds the source device's Sink-ID to its Remote Sink Registry.


Source
Sink
Successful RTT-AKE Procedure
RA_REGISTER(Sink-ID1, RASRC).CMD
Sink's Sink-ID (Sink-ID1) and
Availability of mutual registration
Sink-ID1 is equal to
Device ID or ID_U
in RTT-AKE
N
Y
Sink-ID1 exists in Remote
Sink Registry
Y
N
Remote Sink Registry is full
Y
N
REJECTED().RSP
Abort
Add Sink-ID1 to
Remote Sink Registry
Mutual Registration is possible?
N
Y
Trigger mutual registration in
RA_REGISTER.RSP
ACCEPTED(MREG, Sink-ID2).RSP
If triggered, mutual registration request
and Source's Sink-ID (Sink-ID2)
Mutual Registration is
Requested
N
Y
Sink-ID2 is equal to
Device ID or ID_U in RTT-AKE
N
Y
Abort
Add Sink-ID2 to
Remote Sink Registry

6.2.2 Remote Access AKE (RA-AKE)

Remote access permits sink devices that are listed on the Remote Sink Registry to establish a remote connection to a specific source device if the source device has an unused remote access connection. A Remote Access Connection Registry (RAC Registry) is used to manage each RAC record which consists of Sink-ID, corresponding Remote Exchange Key (K_R), and exchange key label. The following diagram depicts the RA-AKE procedure used to establish the remote access connection.



RA-AKE procedure is as follows:

1. Sink device sends CHALLENGE command with the exchange_key field in which the bit for K_R (Remote Exchange Key) is set. If the bit for K_R is not set, source devices shall abort the RA-AKE procedure. The AKE procedure other than RA-AKE may be continued in such a case.
2. Source and sink devices execute the Challenge-Response portion of Authentication where the TTL constraint in Section 4.6.2 is not applied.

3. Source device checks whether the Sink-ID of the sink device is listed in its Remote Sink Registry. If Sink-ID is not listed, then it sends the AKE_CANCEL and aborts the RA-AKE procedure.
4. Source device checks the RAC Registry to determine if a RAC record exists for the given Sink-ID. If a RAC record exists, then the source device uses the K_R and corresponding exchange_key_label in the RAC record and proceeds to Step 8. Source device may renew the values of K_R and exchange_key_label of the RAC record before going to Step 8 if the source device is not transmitting content using K_R.
5. If a RAC record does not exist for the given Sink-ID, the source device will then check the Remote Access Connection Counter (RACC) Value. If RACC is NOT less than $RACC_{MAX}$[17], the source device must send AKE_CANCEL and abort the RA-AKE procedure. RACC is the counter for remote access connections which is initialized to zero when there are no remote access connection.
6. Source device then increments RACC by one.
7. Source device then generates K_R and corresponding exchange_key_label for the K_R, and put them in a RAC record along with the Sink-ID.
8. Source device then sends the Remote Exchange Key (K_R) and corresponding exchange_key_label associated to the Sink-ID to the sink device.
9. Source device that supports RA_MANAGEMENT then starts a keep-alive timer to maintain the K_R and retain the K_R for at least one minute.
10. SRM transmission follows if an update between source device and sink device is indicated.

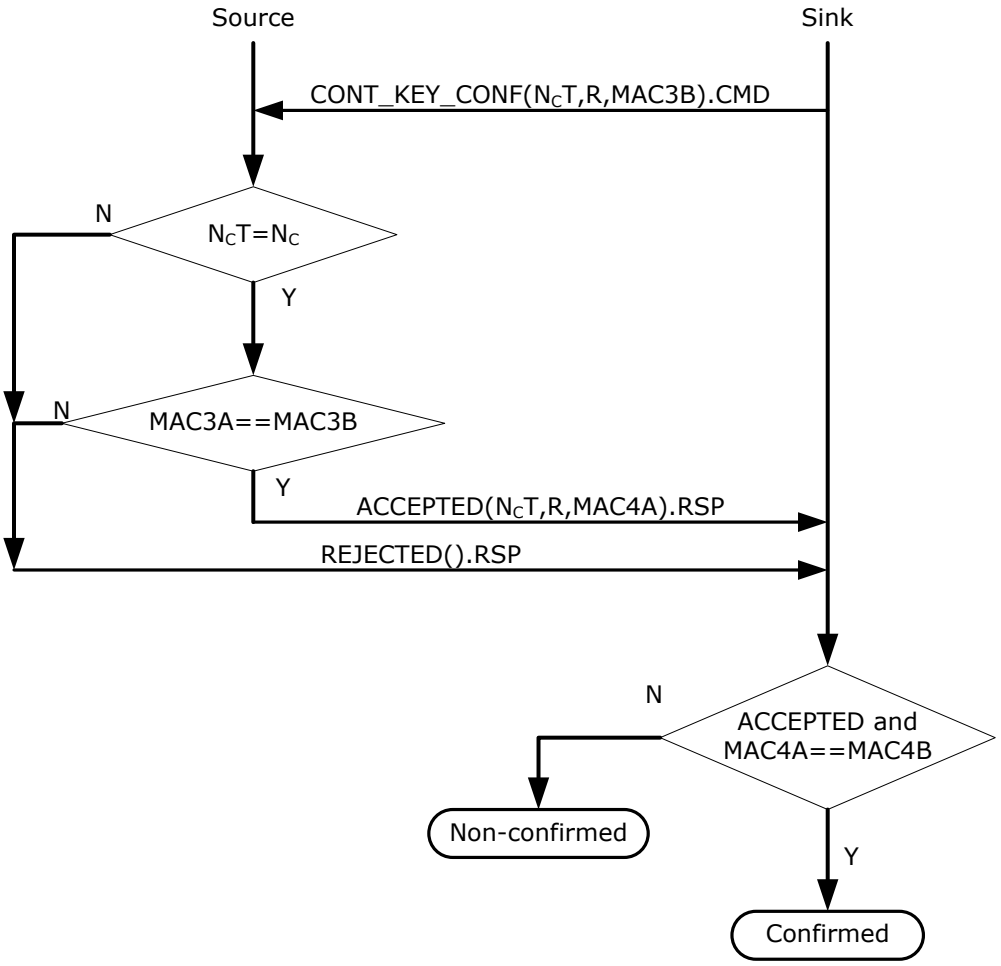
When source device expires a K_R, the source device erases the associated RAC record that contains K_R, and decrements the RACC by one.

6.3 Content Key Confirmation

In the case of multicast content transmission, sink devices shall monitor the N_C value in each PCP containing content encrypted with a Content Key based on a Multicast Exchange Key, and confirm such N_C value is the current one by CONT_KEY_CONF command before starting decryption, then check that N_C value in PCP is incremented by one whenever changes while the next value of $FFFFFFFFFFFFFFFF_{16}$ is zero. When the N_C value changes in other way, sink devices shall immediately terminate decryption for that content stream. When a sink device get an ACCEPTED response for a CONT_KEY_CONF command after a discontinuity of N_C, the sink device may restart decryption of content received after such ACCEPTED response. Sink devices may request N_C value before starting content transmission by CONTENT_KEY_REQ command.

The content key confirmation procedure requires the sink device to send the N_C value under test (N_CT) to the source device. Upon receipt of the N_CT the source device checks such N_CT against its current N_C value being used with the Exchange Key specified by the sink device and if the N_C value equals to the N_CT value then it confirms that N_CT is valid. The confirmation procedure is depicted in following figure.

[17] See Section 6.2.



The functions to calculate MAC values using SHA-256 based on the K_M are described in the DTCP2 Specification available under license from the DTLA.

In the case of unicast content transmission, when a sink device is receiving content stream encrypted with a Content Key based on a Unicast Exchange Key, the above content key confirmation procedure is not needed. However, the sink device shall check that N_C value in PCP is incremented by one whenever changes while the next value of $FFFFFFFFFFFFFFFF_{16}$ is zero. When the N_C value changes in other way, the sink device shall immediately terminate decryption for that content stream. If the discontinuity of N_C happened in a Move transaction (see Section 6.1), the sink device shall abort such Move transaction. The sink device may restart the content transmission session with a new ID_S[18] value.

When the content confirmation procedure is performed for unicast content transmission, in the above figure, K_M is replaced with actual Exchange Key being used, which is either K_S or K_R.

[18] See Section 5.1.4**Error! Reference source not found.**.

7 AKE Command Set

7.1 Introduction

DTCP2 uses TCP Port to send/receive AKE control command packets and status command packets. DTCP2 Socket identification of source device is described in Section 10.2.

7.1.1 AKE Control Command Packet Format

The AKE control command packet format is as follows and is used to exchange commands required to implement the Authentication and Key Exchange protocols. This format is used for both command and response packets.

	msb							lsb
Type[0]	Type = 02_{16}							
Length[0]	(msb) Byte Length of Control fields and AKE_info fields (8+N)							
Length[1]								(lsb)
Control[0]	Reserved (zero)				ctype/response			
Control[1]	Category = 0000_2 (AKE)				AKE_ID = 0000_2			
Control[2]	Subfunction							
Control[3]	AKE_procedure							
Control[4]	exchange_key							
Control[5]	subfunction_dependent							
Control[6]	AKE_label							
Control[7]	Reserved (zero)				Status			
AKE_info[0..N-1]	AKE_info							

Each field of the AKE control command packet is used as follows:

- The **Type** field of the value 02_{16} identifies it as the Version 2 AKE control command packet for DTCP2.
- The **Byte Length** field consists of Length[0] and Length[1] indicates the byte length of the following data. Its value shall be 8+N for either a command packet or a response packet. If there are N AKE_info data bytes to be sent with a command packet, the Byte Length of command shall be 8+N. If there are N AKE_info data bytes to be returned with a response packet, the Byte Length of response shall be 8+N even when the response returns no data.
- The **ctype/response** field has one of the codes defined in the following table to indicate type of command or response. The following name of code concatenated with "response" is used as an abbreviation in this document. (e.g. ACCEPTED response, REJECTED response)

Value	ctype/response	Description
0_{16}	CONTROL	AKE control command
1_{16}	STATUS	Not applicable to AKE control command
2_{16}–7_{16}	Reserved	Reserved for future extension
8_{16}	NOT IMPLEMENTED	Response to AKE control/status command returned if the command is not supported
9_{16}	ACCEPTED	Response to AKE control command returned when the command is accepted
A_{16}	REJECTED	Response to AKE control command returned when the command is rejected
B_{16}	Reserved	Reserved for future extension
C_{16}	STABLE	Not applicable to AKE control Command
D_{16}–F_{16}	Reserved	Reserved for future extension

- The **Category** field of value zero indicates that this packet is for AKE. The values other than zero are reserved.
- The **AKE_ID** field specifies the format of Control[2] through Control[5]. Currently only the encoding AKE_ID = 0_{16} is defined with the format shown in the above. The other values, from 1_{16} to F_{16}, are reserved for future definition.

- The **Subfunction** field specifies the operation of control command. Section 7.1.3 defines the detail of each Subfunction. The name of Subfunction defined in Section 7.1.3 concatenated with "command" and "response" are used as abbreviations of command packet and response packet of that Subfunction in this document. (e.g. CHALLENGE command, CHALLENGE response)
- The **AKE_procedure** field indicates the type of authentication procedure being performed. Each bit of the AKE_procedure field corresponds to one type of authentication procedure. Currently only bit 2 is defined for Baseline_AKE, and the other bits are reserved for future extension and shall be set to zero. In an authentication procedure, initiator of the authentication procedure (sink device) shall set only one bit of this field to specify the type of authentication to be performed, and the same value shall be set to the AKE_procedure field of the following command packets and response packets unless otherwise noted.

Bit	AKE_procedure
0 (lsb) – 1	Reserved for future extension and shall be zero
2	Baseline_AKE procedure
3 – 7 (msb)	Reserved for future extension and shall be zero

- The **exchange_key** field indicates which Exchange Key is shared after authentication. In an authentication procedure, sink device shall set only one bit of this field in the command packet of CHALLENGE subfunction to specify an Exchange Key, and the same value shall be set to the exchange_key field of the following command packets and response packets unless otherwise noted. The following table defines the mapping of Exchange Keys to each bit of the exchange_key field. In the RTT-AKE, bit 3 or bit 5 may only be set to one. In the Move RTT-AKE, only bit 7 may be set to one. In the RA-AKE, only bit 6 may be set to one.

Bit	exchange_key
0 (lsb) – 2	Reserved for future extension and shall be zero
3	Multicast Exchange Key (K_M)
4	Reserved for future extension and shall be zero
5	Session Exchange Key (K_S)
6	Remote Exchange Key (K_R)
7 (msb)	Move Exchange Key (K_{XM})

- The **subfunction_dependent** field has information specific to each subfunction. The definition of the subfunction_dependent field depends on the value of the Subfunction field. Description of each Subfunction in Section 7.1.3 includes how this field is used.
- The **AKE_label** field is a unique tag which is used to distinguish a sequence of AKE commands associated with a given authentication procedure. The initiator of an authentication procedure selects a value for the AKE_label. The selected value should be different from other AKE_label values that are currently in use by the device initiating the authentication. The same AKE_label value is used for all control commands associated with a specific authentication procedure between a source device and a sink device unless otherwise noted. The AKE_label and source Socket of each control command should be checked to ensure that it is from another device performing the authentication procedure.
- The **Status** field in a response packet is used to notify the device issuing the command of the reason when the command results in a REJECTED response. A command packet shall always have the value of 1111_2. A response packet shall have one of the values in the following table unless another table for each Subfunction is specified in Section 7.1.3. If another table is specified, the values in that table shall take precedence over the following table. Note that 0000_2 shall be set to the Status field when the Response code is ACCEPTED in the ctype/response field. The undefined values of the Status field are reserved.

Value	Status	Response code
0000_2	No error	ACCEPTED
0111_2	Any other error	REJECTED

The values of Status field in the following table are for testing purposes only. Products shall not return these values, but instead return 0111_2 (Any other error) if these conditions occur.

Value	Status	Response code
1000_2	Incorrect command order (only for test)	REJECTED
1001_2	Authentication failed (only for test)	REJECTED
1010_2	Data field syntax error (only for test)	REJECTED

Detailed description of the usage for each Status encoding is given in Section 7.1.6.

- The **AKE_info** field contains payload data of each subfunction. The format of this field depends on the values of AKE_ID field and the fields in Control[2] through Control[5], and the format may be different between command and response. Not all subfunctions have this field in its command or/and response. For a response packet with Response code of REJECTED in the ctype/response field, it shall have no AKE_info field. If an AKE_info field have a Reserved field, non-zero values in the Reserved field should be ignored.
- The value of each field in a response packet is identical to that of the corresponding command packet except for the ctype/response, Status and AKE_info fields unless otherwise noted.

7.1.2 AKE Status Command Packet Format

The AKE status command packet format is as follows and is used by sink devices to know the status and capability of a source device. This format is used in both command and response packets. Support of the AKE status command by sink device and source device is optional.

	msb							lsb
Type[0]	Type = 02_{16}							
Length[0]	(msb) Byte length = 0008_{16}							
Length[1]								(lsb)
Control[0]	Reserved (Zero)				ctype/response			
Control[1]	Category = 0000_2 (AKE)				AKE_ID = 0000_2			
Control[2]	Subfunction = FF_{16}							
Control[3]	AKE_procedure							
Control[4]	exchange_key							
Control[5]	subfunction_dependent = FF_{16}							
Control[6]	AKE_label = FF_{16}							
Control[7]	1111_2				Status			

The field format of AKE status command packet is the same as that of AKE control command packet, and each field is used as follows:

- The **Type** field of the value 02_{16} identifies it as the Version 2 AKE status command packet of DTCP2.
- The **Byte Length** field consists of Length[0] and Length[1] indicates the byte length of the following data, and the value shall be 0008_{16}.
- The **ctype/response** field has one of the values defined in the following table to indicate type of command or response.

Value	ctype/response	Description
0_{16}	CONTROL	Not applicable to AKE status command
1_{16}	STATUS	AKE status command
2_{16}–7_{16}	Reserved	Reserved for future extension
8_{16}	NOT IMPLEMENTED	Response to AKE control/status command returned if the command is not supported
9_{16}	ACCEPTED	Not applicable to AKE status command
A_{16}	REJECTED	Not applicable to AKE status command
B_{16}	Reserved	Reserved for future extension
C_{16}	STABLE	Response to AKE status command

D_{16}–F_{16}	Reserved	Reserved for future extension

- The **Category** field of the value zero indicates that this packet is for AKE. The other values than zero are reserved.
- The **AKE_ID** field specifies the format of Control[2] through Control[5]. Currently only the encoding AKE_ID = 0_{16} is defined with the format shown in the above. The other values, from 1_{16} to F_{16}, are reserved for future definition.
- The **Subfunction** field for AKE status command/response packet shall be FF_{16}.
- The **AKE_procedure** field in AKE status command packets shall be set to FF_{16} by sink devices. Source device shall set a value which indicates all of the authentication procedure(s) the source device supports in AKE status response packet. Currently only bit 2 is defined for Authentication, and source devices shall set 04_{16} to the AKE_procedure field.

Bit	AKE_procedure
0 (lsb) – 1	Reserved for future extension and shall be zero
2	Authentication procedure
3 – 7 (msb)	Reserved for future extension and shall be zero

- The **exchange_key** field of AKE status command packets shall be set to FF_{16} by sink devices. Source device shall set a value which indicates all of the Exchange Key(s) the source device supports in AKE status response packet. The following table defines the mapping of Exchange Keys to each bit of the exchange_key field. If a source device supports Session Exchange Key and Remote Exchange Key, the value of exchange_key field in AKE status response packet must be 60_{16}.

Bit	exchange_key
0 (lsb) – 2	Reserved for future extension and shall be zero
3	Multicast Exchange Key (K_M)
4	Reserved for future extension and shall be zero
5	Session Exchange Key (K_S)
6	Remote Exchange Key (K_R)
7 (msb)	Move Exchange Key (K_{XM})

- The **subfunction_dependent** field in AKE status command/response packet shall be FF_{16}.
- The **AKE_label** field in AKE status command/response packet shall be FF_{16}.
- The **Status** field in a AKE status response packet is used to notify current status of a device as a source device to the device issuing the AKE status command. A command packet shall have the value of this field to 1111_2 in the Status field. A response packet shall have one of the values in the following table. The undefined values of the Status field in the following table are reserved. Devices with Source Function set a Status value which is the same one returned when a CHALLENGE command is received, and devices without Source Function always set 0000_2 to the Status field in AKE status response.

Value	Status	Response code
0000_2	No error	STABLE
0001_2	Support for no more authentication procedures is currently available	STABLE
0111_2	Any other error	STABLE

- The value of each field in a response packet is identical to that of the corresponding command packet except for the ctype/response, AKE_procedure, exchange_key, and Status fields.

7.1.3 Subfunction Descriptions

This section describes the format of the subfunctions used to implement the authentication and key exchange protocols. Note that the following sections include descriptions of the fields defined in Section7.1.1. The detailed formats of each subfunction are described in the DTCP2 Specification available under license from the DTLA.

The following table lists currently defined subfunctions with the values of Subfunction field and name of the Subfunction:

Value	Subfunction	Comments
01_{16}	CHALLENGE	Send random challenge and device certificate. This subfunction from a sink device initiates an AKE procedure.
02_{16}	RESPONSE	Return data computed with the received random challenge.
03_{16}	EXCHANGE_KEY	Send a scrambled Exchange Key to the authenticated sink device.
04_{16}	SRM	Send SRM to a device that has an outdated or smaller SRM.
$C0_{16}$	AKE_CANCEL	Notify a device that the current authentication procedure cannot be continued.
84_{16}	CONTENT_KEY_REQ	Confirm availability of an Exchange Key
91_{16}	RTT_READY	Request to setup the RTT measurement.
11_{16}	RTT_SETUP	Request to setup the MAC value for RTT measurement.
12_{16}	RTT_TEST	Request to test RTT.
92_{16}	RTT_VERIFY	Request to verify the MAC value in the last RTT_TEST subfunction.
13_{16}	CONT_KEY_CONF	Request to confirm that N_C value is current one
83_{16}	CMI_REQ	Request to send the latest CMI
$A0_{16}$	MV_INITIATE	Request to start a Move Transaction
21_{16}	MV_EXCHANGE_KEY	Send a scrambled Move Exchange Key (K_{SXM})
28_{16}	MV_CANCEL	Request to cancel a Move Transaction during content transmission
22_{16}	MV_FINALIZE	Request to start Move Commitment process in Move Transaction
23_{16}	MV_COMPLETE	Request to complete Move Commitment process in Move Transaction
31_{16}	RA_REGISTER	Request to register a Sink-ID for Remote Access
32_{16}	RA_MANAGEMENT	Request to keep/expire a Remote Exchange Key (K_R)

7.1.4 Additional AKE Command Rules

7.1.4.1 Cancellation of RTT Procedure

Sink devices may abort RTT procedure by sending REJECTED response to RTT_SETUP, RTT_TEST or RTT_VERIFY subfunction. When a sink device aborts a RTT procedure using the REJECTED response, the source device aborts the RTT procedure that is in progress. When a source device aborts the RTT procedure, the source device sends the AKE_CANCEL subfunction. Sink devices may also use the AKE_CANCEL subfunction to abort RTT procedure.

Source and sink devices shall return REJECTED response with the Status value of 0111_2 and abort RTT procedure upon receipt of a duplicate or an out of sequence command.

If the TCP connection is broken during the RTT procedure, both source and sink devices shall immediately abort the RTT procedure.

7.1.5 Rules for a NOT_IMPLEMENTED response to an AKE Control/Status command

When a device does not support an AKE control/status command identified by Control[0] through Control[7], it should return a response packet to the sender of that command with the response type of NOT_IMPLEMENTED in the ctype/response field. And the NOT_IMPLEMENTED response shall have the

same AKE_info field as that of corresponding command packet. A syntax error in the AKE_info field should not cause a NOT_IMPLEMENTED response.

Following things shall also be considered:

1. The value of the AKE_label field should not cause a NOT_IMPLEMENTED response since the field can have any value.

2. When a Reserved field has a value other than zero, a NOT_IMPLEMENTED response should be returned.

7.1.6 Additional Description of the Status Field

7.1.6.1 Rules for a REJECTED/STABLE response to an AKE Control/Status command

When a device returns a response packet with the response code of REJECTED in the ctype/response field to an AKE control command or AKE status command, it shall set one of the following code in the Status field to specify the reason why the control command was rejected:

Support for no more authentication procedures is currently available (Status = 0001_2)

This Status is used when a source device is requested to start an additional authentication procedure, which it does not currently have the capability to perform. Additionally this status can be used by a device that can act simultaneously as a sink and source, when it receives an authentication request from another device in the middle of performing a separate authentication procedure as a sink device. The sink device that received this response should wait before requesting another authentication procedure.

Any other error (Status = 0111_2)

Both source and sink devices use this code in order to indicate other errors which are not assigned specific codes. For example, this code may be used in the following cases:

- A sink device wants to stop an authentication procedure because it stops receiving content from a source device.
- A device detects an error in the verification of signature or MAC value in an AKE command.
- A device receives a subfunction which is not specified time to send in Section 7.4 when it is not ready to respond.

Note: that the following codes are for testing and debug purposes only and shall not be used in any products. Products should use the **Any other error** code instead.

Incorrect command order (Status = 1000_2) FOR TESTING AND DEBUG ONLY

When a device receives an AKE command that should not be received at that time, the device should return this code. Both source and sink devices may use this code.

Authentication failed (Status = 1001_2) FOR TESTING AND DEBUG ONLY

This code is used when a device determines that the other device cannot be successfully authenticated. Both source and sink devices may use this code.

AKE_info field syntax error (Status = 1010_2) FOR TESTING AND DEBUG ONLY

If the AKE_info field has a syntax error then this code should be used.

7.2 TCP Connection Behavior

If the TCP connection is broken during authentication procedure, both source and sink devices shall immediately abort such authentication procedure.

7.3 Action when Unauthorized Device is detected during Authentication

Devices may return ACCEPTED response before checking validity of the data in the AKE info field of an AKE command, such as verification of signatures in CHALLENGE command and RESPONSE command. However, the devices shall immediately abort the AKE procedure when such checking results in fail.

7.4 Sequence Diagrams

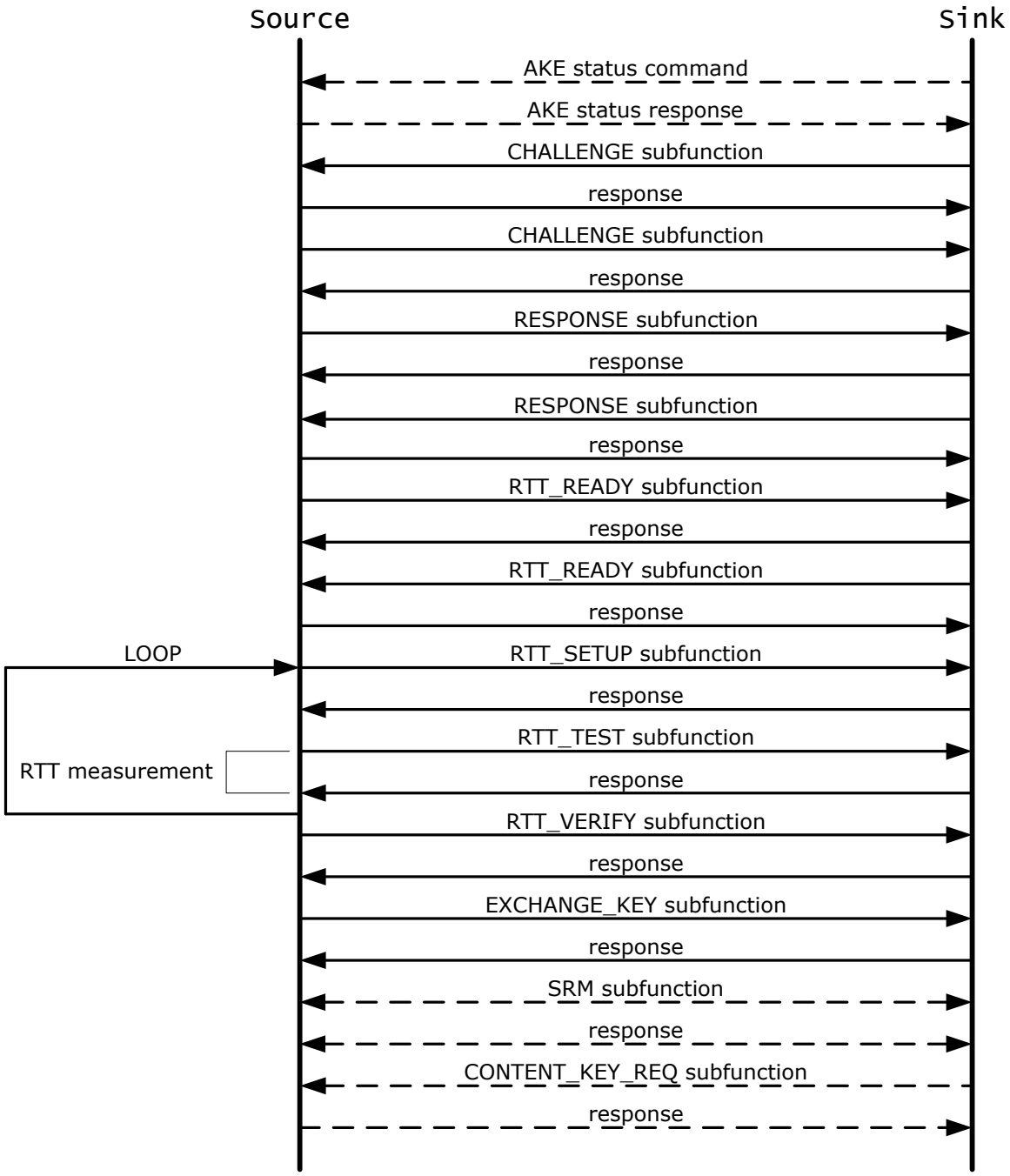
The following figures illustrate the command flows used for AKE.

Notation for sequence diagrams:

- Solid lines indicate command/response pairs that are always performed.
- Dashed lines indicate command/response pairs that are performed on a conditional basis.

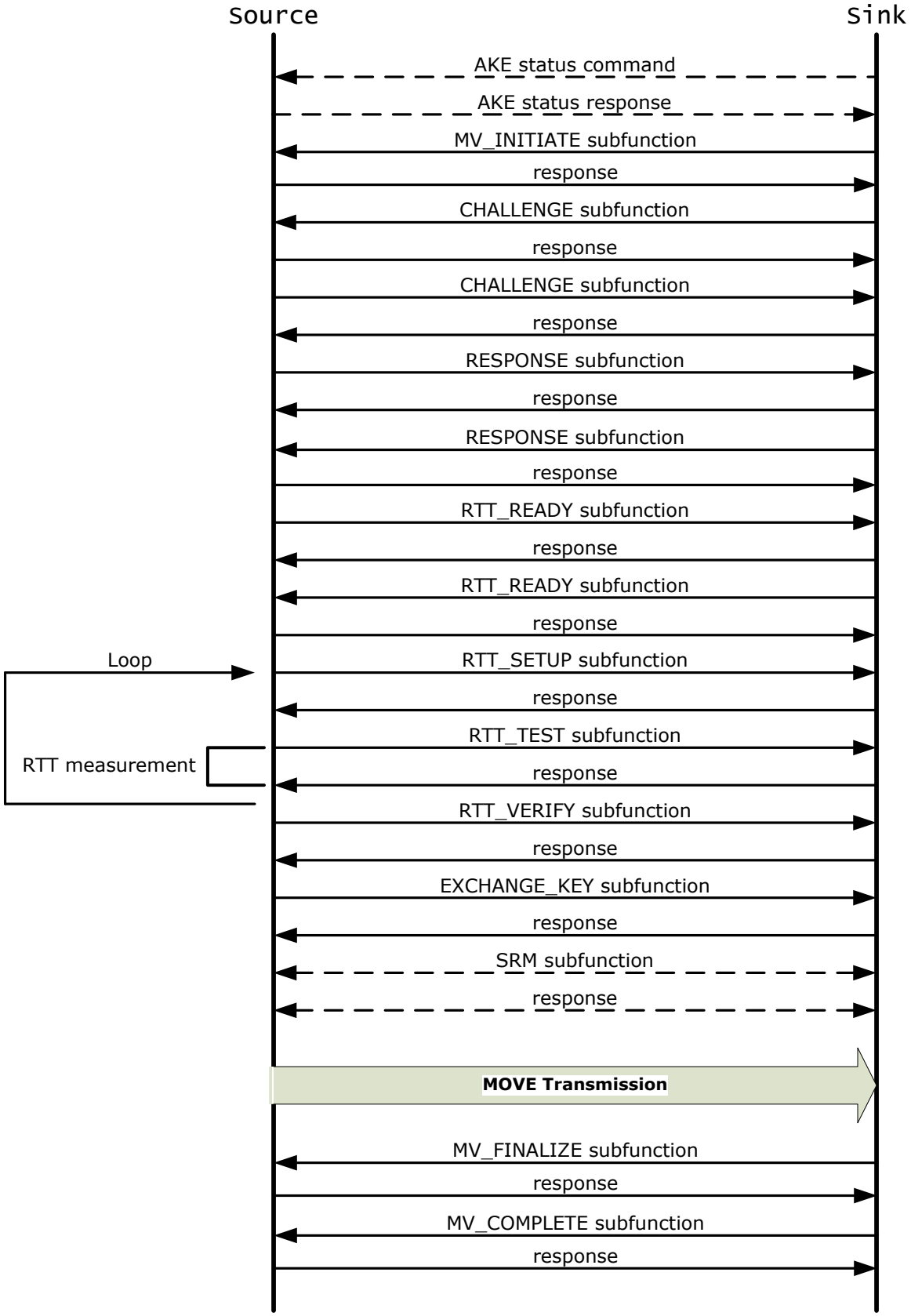
7.4.1 RTT-AKE Sequence

Note that commands and responses for RTT testing (RTT_READY to RTT_VERIFY) are skipped when sink device is registered in source device's RTT registry.



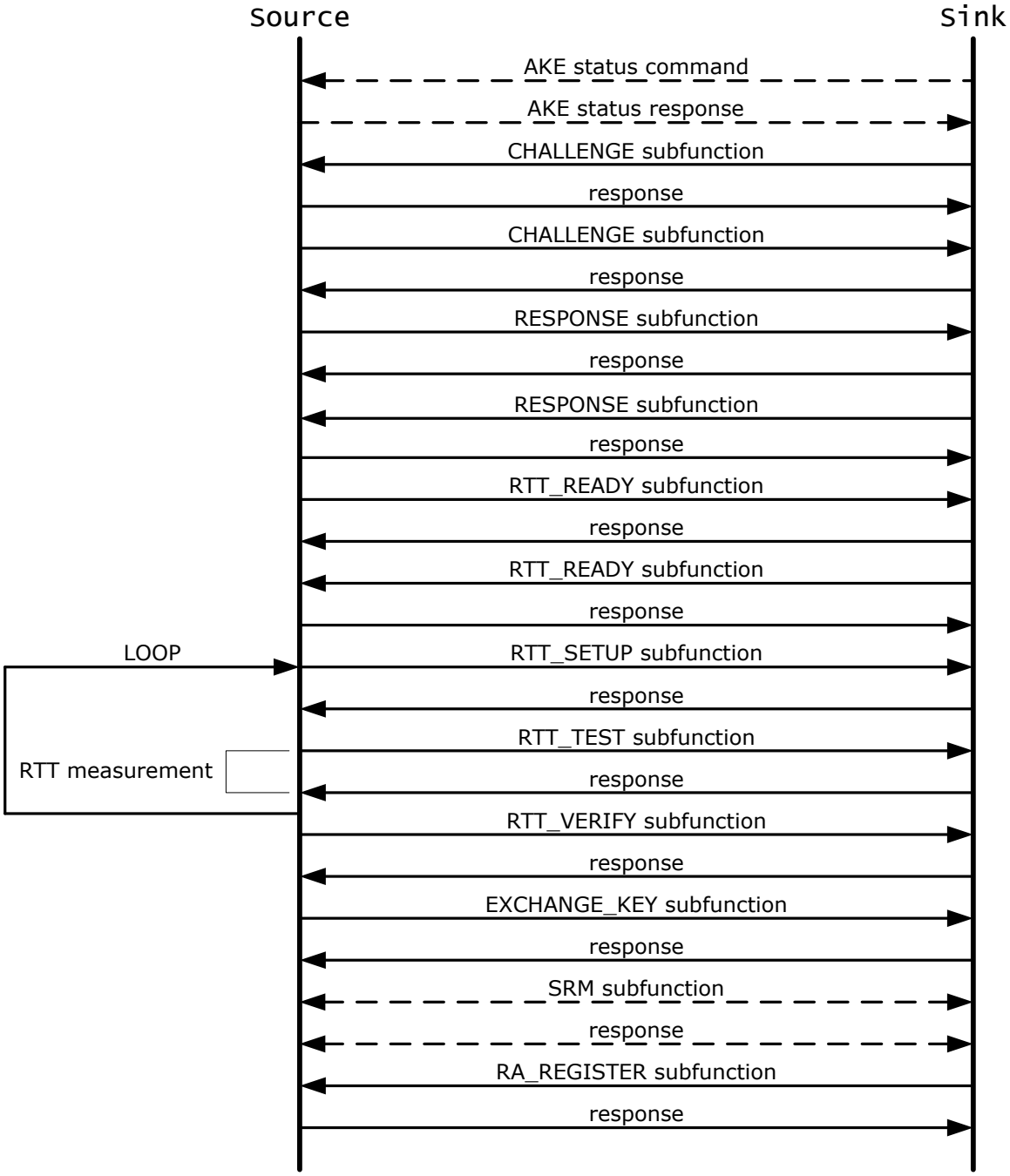
7.4.2 Move RTT-AKE Sequence

Note that commands and responses for RTT testing (RTT_READY to RTT_VERIFY) are skipped when sink device is registered in source device's RTT registry.

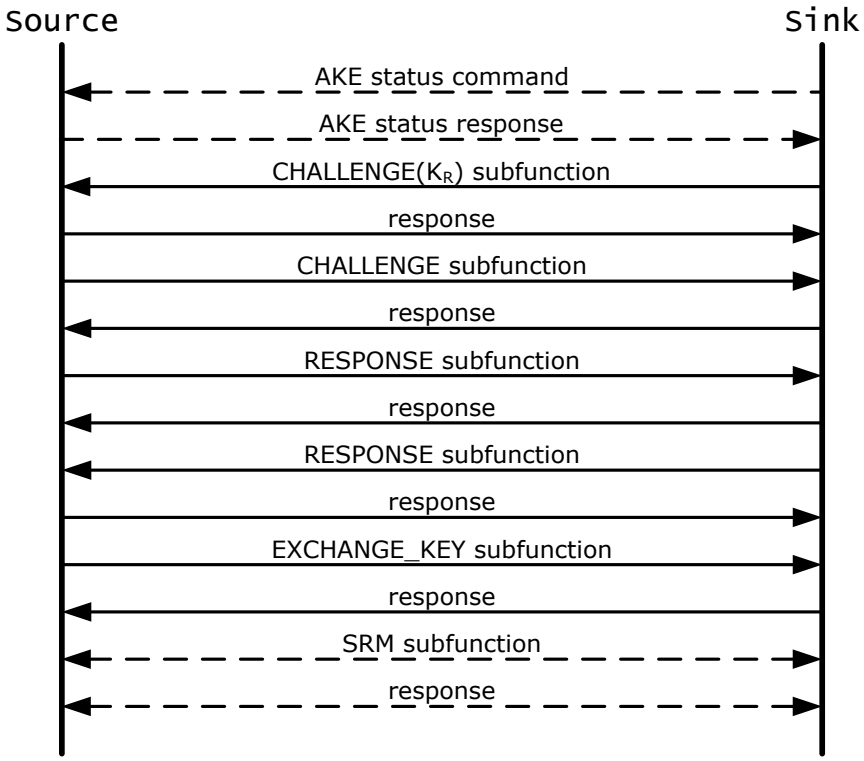


7.4.3 Remote Sink Registration Sequence

Note that commands and responses for RTT testing (RTT_READY to RTT_VERIFY) are skipped when sink device is registered in source device's RTT registry.



7.4.4 RA-AKE Sequence

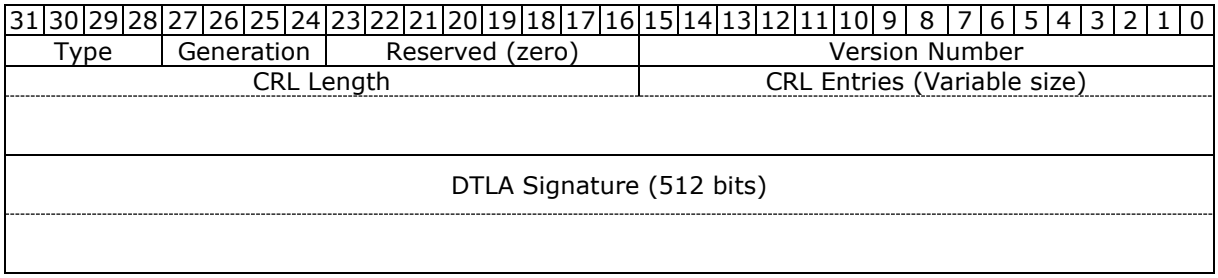


8 System Renewability

Compliant devices can receive and process System Renewability Messages (SRMs) created by the DTLA and distributed with content. These messages are used to ensure the long-term integrity of the system.

8.1 SRM Components and Layout

The structure of First-generation SRMs is shown in the following figure. The fields in the first 4 bytes of the SRM comprise the SRM Header.



Each field of the SRM is used as follows:

- SRM **Type** field (4 bits). This field has the same encoding as is used for the Certificate Type field in the DTCP2 Device Certificate (see Section 3.1.3).
- SRM **Generation** field (SRMM) (4 bits). This field specifies the generation of the SRM. It is used to ensure the extensibility of the SRM mechanism. Currently, the only encoding defined is 0:
    - A value of 0 indicates a First-Generation SRM (Maximum of 16KB).

  Other encodings are currently reserved. The Generation field value corresponds to entire size of the SRM. Note that there may be devices that store part of a newer SRM after a subsequent generation of SRM is defined (e.g. X_{SRMC} <= SRMM. See Section 4.2.3 for X_{SRMC}).
- **Reserved** field (8 bits). These bits are reserved for future definition and are currently defined to have a value of zero.
- SRM **Version Number** (SRMV) (16 bits) which is monotonically increased. This value is exchanged as X_{SRMV} during Authentication. This value is not reset to zero when a new value of message Generation field is defined.
- **CRL Length** field (16 bits). This field specifies the size (in bytes) of the following fields CRL Length field (two bytes), CRL Entries field (variable length), and DTLA Signature field (64 bytes).
- **CRL Entries** field (variable sized). The CRL used to revoke the certificates of devices and DTCP2 implementations whose security have been compromised. Its format is described in the following section.
- The **DTLA Signature** field contains EC-DSA signature for the above components using DTCP2 Device CA Private Key (L^{-1}) (512 bits).

8.1.1 Certificate Revocation List (CRL)

The **Certificate Revocation List (CRL)** identifies devices that are no longer compliant. It can contain device certificate IDs and Implementation IDs. It begins with the CRL Length field that specifies the length of the CRL in bytes. This field is followed by a sequence of Entry Type Block (1 byte) and entries specified by the Entry Type Block. The format of the Entry Type Block is as follows:

7	6	5	4	3	2	1	0
Type			Number of Entries				

The **Type** field indicates the one of the defined CRL types below.

- 0: Device ID
- 1: Contiguous range of Device IDs
- 2: Implementation ID
- 3: Contiguous range of Implementation IDs

The **Number of Entries** field indicates how many entries of that type follow the Entry Type Block. The size if of each entry depends on the value of the Type field as follows:

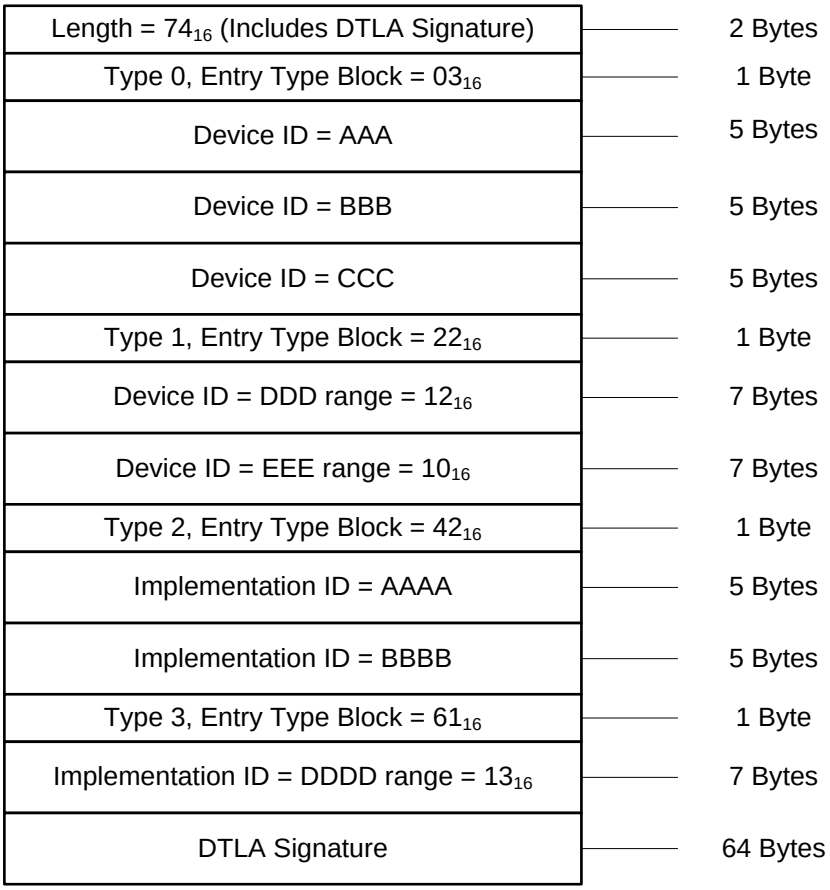
- When Type field value is 0, the size of each entry is 5 bytes, the size of a Device ID
- When Type field value is 1, the size of each entry is 7 bytes, Device ID (5 bytes) and the number of following contiguous Device IDs (2 bytes). The number of contiguous Device IDs is the number of revoked Device IDs that follow the specified Device ID. So when the number of contiguous Device IDs has a value of zero it means only the specified Device ID is revoked.
- When Type field value is 2, the size of each entry is 5 bytes, the size of an Implementation ID.
- When Type field value is 3, the size of each entry is 7 bytes, Implementation ID (5 bytes) and the number of following contiguous Implementation IDs (2 bytes). The number of contiguous Implementation IDs is the number of revoked Implementation IDs that follow the specified Implementation ID. So when the number of contiguous Implementation IDs has a value of zero it means only the specified Implementation ID is revoked.

Note:

- That the end of the CRL Entries field is padded with 0 to 3 bytes of 00_{16} to obtain 32-bit alignment.
- That Type 1 and Type 3 have the same syntax.

The following is an example CRL:

- revoking Device IDs AAA, BBB, CCC, DDD–(DDD+18), and EEE–(EEE+16)
- revoking Implementation IDs, AAAA, BBBB, and DDDD-(DDDD+19)

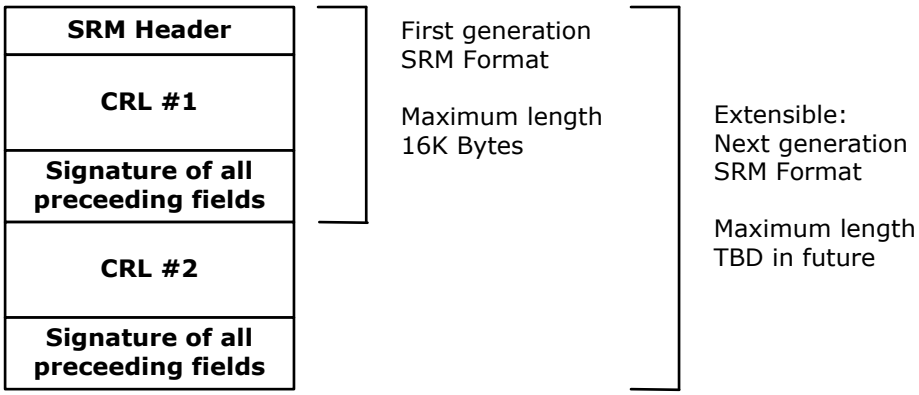


8.1.2 DTLA EC-DSA Signature

The DTLA Signature field is a 512-bit signature calculated over all of the preceding fields of the SRM using the DTCP2 Device CA Private Key (L^{-1}). This field is used to verify the integrity of the SRM using the DTCP2 Device CA Public Key (L^{1}).

8.1.3 SRM Scalability

To ensure the scalability of this renewability solution, the SRM format is extensible. Next-generation extensions (CRLs and possibly other mechanisms) to a current-generation SRM format must be appended to the current-generation SRM (as shown in following figure) in order to ensure backward compatibility with devices that only support current/previous-generation SRMs. Devices are only responsible for supporting the generation of SRM that was required by the DTLA as of the time the device was manufactured. The conditions under which the DTLA will authorize a new-generation SRM are specified in the DTCP2 Digital Transmission Protection License Agreement.



8.1.4 Device to Device Update and State Machine

8.1.4.1 Updating a Device's SRM from Another Compliant Device

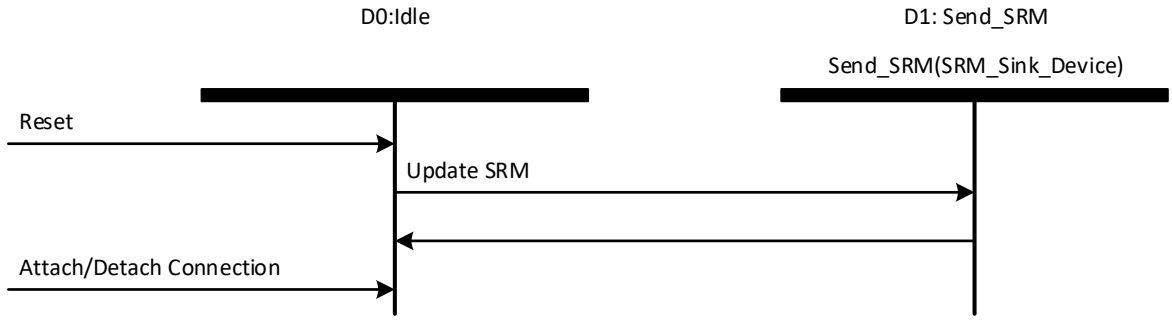
During the Authentication procedure, if a more recent (or more complete) system renewability message is discovered on another device, the following procedure is used to update the device with the outdated and/or less complete copy:

1. The device with the newer and/or more complete SRM sends it to the other device.
2. The device being updated verifies the candidate SRM's signature with the DTCP2 Device CA Public Key.
3. If the signature is valid, the device being updated replaces the entire currently stored SRM in its non-volatile storage with as much of the replacement message as will fit in its non-volatile storage.

This procedure should take place following the completion of the exchange of Exchange Key.

8.1.4.2 System Renewability State Machine (Device-to-Device)

The following figure depicts the *SRM Exchange_Source* State Machine showing the exchange of SRMs between devices from the point of view of the device that is the source of the SRM. The detail of this state machine is described in the DTCP2 Specification available under license from the DTLA.



8.1.5 Receipt of DTCP2 SRM via Alternate Distribution Path

DTCP2 SRMs may be distributed via files and included in a communication interface, such as broadcast streams and package media where an entity is required via agreement to deliver DTCP2 SRMs to the Source Function. In the case the Source Function shall perform the following checks before accepting the SRM.

- Verify that the SRM version (SRMV) is greater than the one currently held by the Source Function, or the SRM version is the same and the SRMM of the candidate SRM is greater than the X_{SRMC} of the Source Function. Otherwise, the Source Function shall not accept that SRM.

- The Source Function verifies the candidate SRM's signature with the DTCP2 Device CA Public Key. If the signature is not valid, the Source Function shall not accept that SRM.
- The Source Function will replace the entire currently stored SRM in its non-volatile storage with as much of the replacement message as will fit in its non-volatile storage where the replaced message ends with a DTLA signature (i.e. If the SRMM of the candidate SRM is greater than the X_{SRMC} of the Source Function, the candidate SRM is stored from the first generation to the end of the generation of that X_{SRMC}).

9 Limitation of the Number of Sink Devices Receiving Content

Without exception, the number of authenticated sink devices by a source device, including sink devices connected via Bridge Devices that can decrypt content from that source device shall be limited to no more than 34 devices at any time. For avoidance of doubt, this requirement is not applied to Bridge Devices as source devices.

9.1 Limitation Mechanism in Source Device

Source device shall count the number of sink devices that can decrypt content from that source device using a counter (Sink Counter). The source device shall increment the Sink Counter and register the sink device's Device ID for Unique-key Device or ID_U for Common-key Device to a list (Sink List) after a successful AKE[19] with an unregistered sink device that has a device certificate with AP flag value of zero (Non-Bridge Sink). The source device shall also increment the Sink Counter after a successful AKE regardless of its registration status when a sink device has a device certificate with AP flag value of one (Bridge Device). If the source device outputs content through multiple interfaces, it shall count the number of sink devices on those interfaces using one Sink Counter. Note that the above requirements shall be fulfilled regardless of the type of AKE (e.g. RTT-AKE, Move RTT-AKE and RA-AKE) specified in this Specification.

Except for the following cases, source devices shall neither decrease its Sink Counter nor unregister any sink devices from the Sink List.

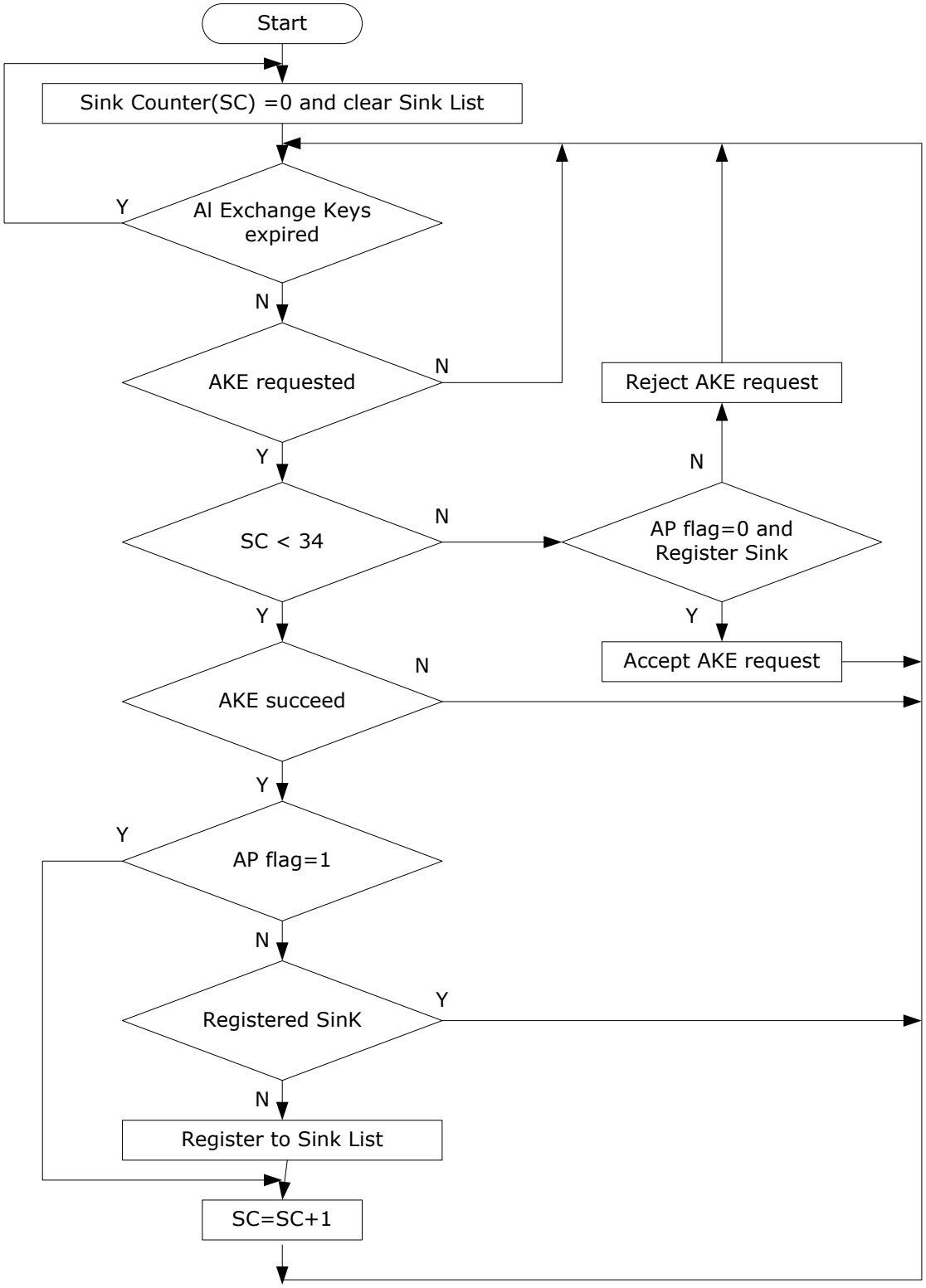
- When a source device expires all provided Exchange Keys (K_M, K_S, K_R and K_{XM}), it shall reset the Sink Counter to zero and clear the Sink List.
- When a source device expires an Exchange Key, if the source device can identify Non-Bridge Sinks that have no valid (unexpired) Exchange Key provided from that source device, it may decrement the Sink Counter by the number of such Non-Bridge Sinks by deleting registered Device IDs and ID_Us of such devices from the Sink List.
- When a source device expires an Exchange Key, if the source device can identify Bridge Devices to which such Exchange Key was provided and also identify the number of successful AKEs to provide such Exchange Key to Bridge Devices, the source device may decrement the Sink Counter by that number of successful AKEs.

Source devices should not expire an Exchange Key (K_M or K_S) which has been provided to a Bridge Device except when its Sink Counter is reset to zero by the expiration of such Exchange Key, and provide the same value of Exchange Key and exchange_key_label whenever the Bridge Device requests further authentication. However, if a source device can identify valid Exchange Keys provided to a Bridge Device and the number of successful AKEs between such Bridge Device for such Exchange Keys, the source device may expire such Exchange Keys by decrementing the Sink Counter by that number of successful AKEs. When source devices expire an Exchange Key provided to a Bridge Device, all Exchange Keys provided to such Bridge Device should be expired at the same time.

When the Sink Counter reaches the maximum limit (34 or less), the source device shall reject any further authentication requests from both unregistered Non-Bridge Sinks and Bridge Devices by returning REJECTED response with the Status code of 0111_2 (Any other error) to CHALLENGE command. This Status code should not be used for other commands to indicate that the Sink Counter reaches the maximum limit except for the AKE status command.

The following figure shows a sample flow of the limitation mechanism by source devices (SC means Sink Counter):

[19] After Successful AKE means after source device sends an Exchange Key.


Start
Sink Counter(SC) =0 and clear Sink List
AI Exchange Keys expired
Y
N
AKE requested
N
Reject AKE request
Y
N
SC < 34
N
AP flag=0 and Register Sink
Y
Y
Accept AKE request
AKE succeed
N
Y
AP flag=1
Y
N
Registered SinK
Y
N
Register to Sink List
SC=SC+1

9.2 Limitation Mechanism in Bridge Device

Bridge Device is a device having both a sink function and a Source Function where the sink function decrypts content received from one or more upstream source devices and the Source Function simultaneously encrypts and transmits such decrypted content to one or more downstream sink devices. For avoidance of doubt, the Source Function of Bridge Device shall generate Exchange Key(s) and use it for encryption of content sent to downstream sink devices, where such Exchange Key shall be either Session Exchange Key (K_S) or Multicast Exchange Key (K_M).

Before providing an Exchange Key for decryption of content to a downstream sink device, Bridge Device shall complete a successful AKE between the upstream source device of such content using a device certificate with AP flag of 1 unless such sink device is registered in the Sink List of the Bridge Device and is using a device certificate with AP flag of 0 (Non-Bridge Sink). When the sink device is a registered Non-Bridge Sink, the Bridge Device shall provide the Exchange Key without AKE between such upstream source device.

When an AKE between an unregistered Non-Bridge Sink is successfully completed, Bridge Devices shall register the Device ID for Unique-key Device or ID_U for Common-key Device of such Non-Bridge Sink to its Sink List.

Exchange Key obtained from an upstream source device by an authentication using a device certificate with AP flag of one shall not be used other than transcrypting use for bridging in the Bridge Device.

9.2.1 Bridging Only One Source Device

If Bridge Device only transmits content from a single upstream device (Content Source) which is a source device or another Bridge Device to downstream sink devices, the followings shall be satisfied:

- When Bridge Device receives an AKE request from an unregistered Non-Bridge Sink or a downstream Bridge Device, the Bridge Device shall complete a successful AKE between the Content Source before accepting such AKE request. If the Bridge device rejects the AKE request to make the sink device wait for completion of AKE between the Content Source, it should use the Status code of 0001_2"Support for no more authentication procedure is currently available" for REJECTED response to CHALLENGE command. However, if the AKE between the Content Source is rejected with the Status code of 0111_2 "Any other error", this Status code should be used.
  Bridge Device may complete one advanced AKE between the Content Source before receiving next AKE request from a downstream sink device to accept the next AKE request immediately. Note that it is strongly recommended not to perform two or more advanced AKEs between Content Source.
- When Bridge Device detects that an Exchange Key provided by the Content Source was expired, the Bridge Device shall clear its Sink List. Bridge Devices are strongly recommended not to clear its Sink List in other conditions to avoid unnecessary increment of the Sink Counter in the Content Source. With the same reason, Bridge Devices are also strongly recommended not to expire Exchange Key and corresponding exchange_key_label provided by a Content Source except when the Bridge Devices detect that the Content Source expired such Exchange Key.
- Bridge Devices should not expire an Exchange Key (K_M or K_S) which has been provided to a downstream Bridge Device except when its Sink List is cleared, and provide the same value of Exchange Key and exchange_key_label whenever the downstream Bridge Device requests further authentication. When Bridge Devices expire an Exchange Key provided to a downstream Bridge Device, all Exchange Keys provided to such Bridge Device should be expired at the same time.

9.2.2 Bridging Two or More Source Devices

If Bridge Device transmits content from two or more upstream devices (Content Sources) which are a source devices and/or another Bridge Devices to downstream sink devices, the followings shall be satisfied:

- When Bridge Device receives an AKE request from an unregistered Non-Bridge Sink or a downstream Bridge Device, the Bridge Device shall complete successful AKEs between all of the Content Sources before accepting such AKE request. If the Bridge Device rejects the AKE request to make the sink device wait for completion of AKEs between the Content Sources, it should use the Status code of 0001_2"Support for no more authentication procedure is currently available" for REJECTED response to

CHALLENGE command. However, if an AKE between a Content Source is rejected with the Status code of 0111_2 "Any other error", this Status code should be used.
Bridge Device may complete one advanced AKE between each Content Source before receiving next AKE request from a downstream sink device to accept the next AKE request immediately. Note that it is strongly recommended not to perform two or more advanced AKEs between a Content Source.

- Bridge Device shall have a Sink Counter for each Content Source and increment the Sink Counter corresponding to a Content Source by one when an AKE between such Content Source is succeeded.
- When the Sink Counter value for Content Source-X is less than a Sink Counter value for another Content Source, AKE requests from downstream sink devices other than Non-Bridge Devices registered in the Sink List shall be rejected. Otherwise, may be accepted under the condition that content from the Content Source-X is not transmitted to any downstream sink devices.
- When the Sink Counter value for Content Source-X is less than a maximum Sink Counter value for another Content Source, Bridge Device may perform AKEs between the Content Source-X until the Sink Counter value reaches to the largest Sink Counter value for another Content Source.
- When Bridge Device detects that an Exchange Key provided by a Content Source was expired, the Bridge Device shall reset the Sink Counter corresponds to such Content Source to zero. When all of the Sink Counters for Content Sources become zeros, Bridge Device shall clear its Sink List. Bridge Devices are strongly recommended not to clear its Sink List and not to reset its Sink Counter to zero in other conditions to avoid unnecessary increment of the Sink Counter in any Content Sources. With the same reason, Bridge Devices are also strongly recommended not to expire Exchange Key and corresponding exchange_key_label provided by a Content Source except when the Bridge Devices detect that the Content Source expired such Exchange Key.
- Bridge Devices should not expire an Exchange Key (K_M or K_S) which has been provided to a downstream Bridge Device except when its Sink List is cleared, and provide the same value of Exchange Key and exchange_key_label whenever the downstream Bridge Device requests further authentication. When Bridge Devices expire an Exchange Key provided to a downstream Bridge Device, all Exchange Keys provided to such Bridge Device should be expired at the same time.
- Bridge Device shall use the same Exchange Key (K_S or K_M) for downstream transmissions regardless Content Source.

9.3 Additional Device Certificate in a Bridge Device

Bridge Device may have device certificate with the AP flag value of zero in addition to the device certificate with the AP flag value of one. The device ID of these two device certificates are different each other.

Exchange Key obtained by an authentication using a device certificate with AP flag of zero shall be used for a function in a sink device independent of transcrypting use for bridging in the Bridge Device, such as rendering and recording of received content like sink devices other than Bridge Device, or such successful authentication shall be treated as one advanced AKE in 9.2.1 or 9.2.2 regardless of the times the Bridge Device obtains the same Exchange Key.

Bridge Device may use the device certificate with AP flag of zero for a Source Function independent of transcrypting use. In such a case, the Source Function shall count the number of downstream sink devices that can decrypt content from the Source Function according to the rules described in Section 9.1.

10 Recommendations

The followings are recommendations to ensure interoperability among devices implementing DTCP2. If any of the followings is implemented, such implementation shall be fully compliant to the specification of the corresponding section. For example, if the DTCP2.COM_FLAGS param is implemented, all of the defined fields of the DTCP2.COM_FLAGS param in Section 10.5.1 shall have corresponding values to the associated content.

10.1 Recommended MIME type for DTCP2 Protected content

The DTCP2 application media type is as follows:

application/x-dtcp2; CONTENTFORMAT=<mimetype>

Where **CONTENTFORMAT**, is the standard content media type that is protected by DTCP2.

In addition, information identifying a DTCP2 Socket may be included as follows:

application/x-dtcp2; DTCP2HOST=<host>; DTCP2PORT=<port>; CONTENTFORMAT=<mimetype>

Where:

DTCP2HOST specifies the IP address and **DTCP2PORT** specifies the port number of the DTCP2 Socket of a source device.

In the case of content applicable to Remote access, information may be added as follows:

application/x-dtcp2; DTCP2HOST=<host>; DTCP2PORT=<port>; DTCP2RAPORT=<port>; CONTENTFORMAT=<mimetype>

Where:

The DTCP2 Socket for Remote Access is specified by **DTCP2RAPORT** which is the port number for RA-AKE along with the IP address specified with the **DTCP2HOST**.

Content type of HTTP response / request is set to DTCP2 application media type.

10.2 Identification of DTCP2 Sockets

DTCP2 uses a TCP port to support various commands and control protocols (e.g. RTT-AKE, Move RTT-AKE) and either a TCP or UDP for content transport. This section details recommended practices for identifying DTCP2 Sockets.

10.2.1 URI Recommended Format

This following information is inserted into the query string portion of URI, and is used to provide identifier to access content and DTCP2 Socket of the source device to the sink device. The source obtains the sink's DTCP2 Socket when the sink establishes a TCP connection to the source.

<service>://<host>:<port>/<path>/<FileName>.<FileExtention>?CONTENTPROTECTIONTYPE=DTCP2&DTCP2HOST=<host>&DTCP2PORT=<port>

Where:

CONTENTPROTECTIONTYPE is set to "DTCP2" where 2 represents a DTCP version number that may be incremented in the future as needed.

DTCP2HOST specifies the IP address and **DTCP2PORT** specifies the port number of the DTCP2 Socket of the source device. The DTCP2 Socket for Remote Access is specified by **DTCP2RAPORT** which is the port number for RA-AKE along with the IP address specified with the **DTCP2HOST**.

10.2.2 HTTP response/request

Content type of HTTP response / request[20] is set to the DTCP2 application media type as follows:

Content-Type: application/x-dtcp2; DTCP2HOST=<host>; DTCP2PORT=<port>; CONTENTFORMAT=<mimetype>

10.3 Header Field Definition for HTTP

The following header fields are defined for HTTP transfers.

10.3.1 Range.dtcp.com

The Range.dtcp.com header is used in the same manner as the RANGE header defined in RFC 2616 except that range specification applies to the content before DTCP2 processing.

10.3.2 Content-Range.dtcp.com

The Content-Range.dtcp.com header is used in the same manner as the CONTENT-RANGE header defined in RFC 2616 except that range specification applies to the content before DTCP2 processing.

10.3.3 Session.dtcp.com

The Session.dtcp.com header is used to specify a Session Exchange Key to be used in a content transmission. The exchange_key_label is the parameter of this header as follows:

Session.dtcp.com:<K_S_label>:<ID_{SU}>

<K_S_label> includes the value of the exchange key label of the Session Exchange Key and is denoted by 2 hexadecimal digits. Note that the first digit must be 0 when the value of exchange_key_label is less than 10_{16}.

<ID_{SU}> includes the value of the ID_{SU} in Section 5.1.4 and is denoted by 16 hexadecimal digits.

10.3.4 BLKMove.dtcp.com

The BLKMove.dtcp.com header is used to specify which K_{XM} is used in the Move Transmission process specified in 6.1.3. K_{XM}_label is the parameter of this header as follows:

BLKMove.dtcp.com:<K_{XM}_label>:<ID_{SU}>

<K_{XM}_label> is denoted by 2 hexadecimal digits. Note that the first digit must be 0 when the value of K_{XM}_label is less than 10_{16}.

<ID_{SU}> includes the value of the ID_{SU} in Section 5.1.4 and is denoted by 16 hexadecimal digits.

10.3.5 RemoteAccess.dtcp.com

The RemoteAccess.dtcp.com header is used to specify Remote Exchange Key to be applied for the content transmission through the HTTP transfer including this header field. K_R label is a parameter of this header as follows:

RemoteAccess.dtcp.com:<K_R_label>:<ID_{SU}>

<K_R_label> includes the value of the exchange key label of the Remote Exchange Key and is denoted in hexadecimal 2 digits. Note that the first digit must be 0 when the value of exchange_key_label is less than 10_{16}.

<ID_{SU}> includes the value of the ID_{SU} in Section 5.1.4 and is denoted by 16 hexadecimal digits.

[20] For example, HTTP POST request with “Expect: 100-continue” header.

10.3.6 CopyCount.dtcp.com

The CopyCount.dtcp.com header is used by sink devices to request content transmission based on a Unicast Exchange Key using CMI Descriptor(s) with the CC field. The CC_req flag is the parameter of this header as follows:

CopyCount.dtcp.com:<CC_req>

<CC_req> includes the value specified in Section 5.2.10 and is denoted by 1 hexadecimal digit.

For non Copy Count content, sink devices do not use this header field or set "0" to <CC_req>, and source devices ignore this header field.

10.4 HTTP Status Codes in Error Cases

Source devices should use the following HTTP status codes under the specified conditions:

403 Forbidden:

When source devices receive an HTTP request to transmit content by using a specified Exchange Key, if such Exchange Key has already been expired, this status code should be returned in the HTTP response. Source devices are recommended not to use this code for other purposes in HTTP responses to a sink device.
Sink devices that received this code should expire the Exchange Key specified in the HTTP request.

503 Service Unavailable:

When source devices receive an HTTP request to transmit content by using a specified Exchange Key, if such Exchange Key is valid but content transmission is not possible with some reason, such as necessary resource is currently allocated to other purpose, this status code should be returned in the HTTP response. Source devices should not return this code to the requests which cannot be accepted forever.

10.5 Definition for UPnP AV CDS[21] Property

The following is defined for properties in UPnP AV CDS.

10.5.1 DTCP2.COM_FLAGS param

The DTCP2.COM_FLAGS param is used in the 4th field of res@protocolInfo property to show static attribute of content regarding DTCP2 transmission. Note that the content management information in CMI packet must take precedence over the information in the DTCP2.COM_FLAGS param to handle received content by sink devices (See Section 5.3.1.3.). The DTCP2.COM_FLAGS param is a 32 bit field, and the bit definition is as follows:

Bit 31: DTCP2 Movable
Bit 30: Move protocol is supported
Bit 29: Copy-count carried in CMI
Bit 28: Analog output is restricted
Bit 27: SDO
Bit 26: HDR
Bit 25: L2-Only
Bit 24: EI
Bit 23-4: Reserved (zero)
Bit 3-0 Version of DTCP2.COM_FLAGS param

Bit 31 is set to one if associated content can be moved using DTCP2.

Bit 30 is also set to one if the content can be moved based on the Move protocol in Section 6.1. When only bit 31 is set to one, the Move protocol cannot be used.

Bit 29 is set to one if the CC field has a value more than one when used in Move Transmission.

[21] Refer to UPnP ContentDirectory:2 document.

Bit 28 is set to zero if both AST field and DOT field are fixed to zero during transmission. Otherwise, Bit 28 is set to one.

Bit 27 is set to the identical value to the SDO field in the CMI for that content if that value is constant. If the value of SDO field can change during transmission, Bit 27 is set to zero.

Bit 26 is set to the identical value to the HDR field in the CMI for that content if that value is constant. If the value of HDR field can change during transmission, Bit 26 is set to one.

Bit 25 is set to the identical value to the L2-Only field in the CMI for that content if that value is constant. If the value of L2-Only field can change during transmission, Bit 25 is set to one.

Bit 24 is set to the identical value to the EI field in the CMI for that content if that value is constant. If the value of EI field can change during transmission, Bit 24 is set to one.

Bits 23 to 4 are reserved, set to zero, and are to be ignored.

Bit 3 to Bit 0 shows the version of DTCP2.COM_FLAGS param. The value of these four bits indicates valid bits in the DTCP2.COM_FLAGS that have corresponding values to the associated content as follows:

Version value	Valid bits
0000_2	Bit 31 to Bit 29
0001_2	Bit 31 to Bit 24
0010_2-1111_2	Reserved for future extension

Implementations which use DTCP2.COM_FLAGS param based on this revision of the DTCP2 Specification shall set 0001_2 to these four bits.

The 32 bit value of DTCP2.COM_FLAGS param is denoted by 8 hexadecimal digits.

10.5.2 res@dtcp2:uploadInfo

The res@dtcp2:uploadInfo property is used to show how the content is uploaded using DTCP2. The res@dtcp2:uploadInfo property is a 32 bit field, and bit definition is as follows:

Bit 31: Content will be moved using DTCP2 Move
Bit 30: Move protocol will be used
Bit 29: Copy-count carried in CMI
Bit 28-0: Reserved (zero)

Bit 31 is set to one if associated content will be moved using DTCP2.

Bit 30 is also set to one if the move will be executed based on the Move protocol in Section 6.1. When only bit 31 is set to one, the Move protocol is not used.

Bit 29 is set to one if the CC field has a value more than one when used in Move Transmission.

Reserved bits are set to zero. Devices refer to the reserved bits ignore the value.

The 32 bit value of res@dtcp2:uploadInfo is denoted by 8 hexadecimal digits.

The definition of XML namespace whose prefix is "dtcp2:" is "urn:schemas-dtcp-com:metadata-1-0/dtcp2/".

10.5.3 res@dtcp2:RSRegiSocket

The res@dtcp2:RSRegiSocket property is used to describe one or more DTCP2 Sockets for the Remote Sink Registration. The res@dtcp2:RSRegiSocket property is composed of a comma-separated list of the Sockets, and included in the first item in a CDS:Browse response. If all items in a CDS:Browse response are served by a single UPnP Media Server the res@dtcp2:RSRegiSocket property has only one Socket, otherwise the Socket of each UPnP Media Server is listed in the res@dtcp2:RSRegiSocket property. [e.g. <host1>:<port1>,<host2>:<port2>]

The definition of XML namespace whose prefix is "dtcp2:" is "urn:schemas-dtcp-com:metadata-1-0/dtcp2/".

10.6 Definition for UPnP Device Description

The following is defined for elements in the UPnP Device Description.

10.6.1 dtcp2:X_DTCP2CAP

<dtcp2:X_DTCP2CAP> element (as a child of the <device> element that represents the UPnP AV MediaServer) is used to indicate the support of the following function as a UPnP AV MediaServer with the specified Capability ID:

Function	Capability ID
Upload copy	dtcp2-copy
Upload move	dtcp2-move
Resumption of move	dtcp2-move-resumption

The definition of XML namespace whose prefix is "dtcp2:" is "urn:schemas-dtcp-com:metadata-1-0/dtcp2/".

If two or more of the above functions are supported, Capability IDs are separated with comma as shown in the following example:

```
<dtcp2:X_DTCP2CAP xmlns:dtcp2="urn:schemas-dtcp-com:metadata-1-0/dtcp2/">dtcp2-move,
dtcp2-move-resumption</dtcp2:X_DTCP2CAP>
```

10.7 Recommended Media Profile ID

If there is no corresponding Media Profile ID for content in the DLNA Guidelines, alternative Media Profile ID (apn-param) with the following syntax (EBNF notation) should be included in the 4th field of protocolInfo in the same way as the other-param of the DLNA Guidelines to indicate format of content.

- apn-param = [apn-param-delim] IANA-name "_PN=" apn-value
- [apn-param-delim] = ";"
- IANA-name = <IANA-registered name, with top level domain (e.g. .net, .org, .com)>
- apn-value = [Protect]"_"Container"_"[V-codec *("+"V-attr)]"_"[A-codec *("+"A-attr)]["_"Sup]
- Protect = "DTCP2"
- Container = "TTS" | "MMTTLV" | "MP4"
- V-codec = "HEVC" | "AVC"
- V-attr = "8K" | "4K" | "HD" | "HLG" | "PQ" | "SDR" | "SDR2020" | "25" | "30" | "50" | "60" | "100" | "120"
- A-codec = "M4AAC" | "M4ALS" | "M2AAC"
- A-attr = "MULT5" | " MULT7" | " MULT22"

The format of container, types of video codec and audio codec are indicated by Container, V-codec and A-codec, respectively. If rendering of content requires some sort of video/audio capability(s), such capability(s) is indicated by V-attr(s) and/or A-attr(s). If two or more values of V-attr or A-attr are used in an apn-value, they are unordered.

If format of content may change, the apn-value shows the most demanding condition. For example, if content consists of “HLG” part and “SDR” part, V-attr for such content includes “HLG”.

Sup is TBD, and the values for the above elements will be extended to cover other content format.

The following table shows the reference information for the above values:

Container

Value	Reference
TTS	4-byte timestamp followed by 188-byte MPEG Transport Stream Packet (ISO/IEC 13818-1)
MMTTLV	MMT-based Media Transport Scheme in Digital Broadcasting Systems (ARIB STD-B60)
MP4	ISO base media file format (ISO/IEC 14496-12 or 14496-15)

V-codec

Value	Reference
HEVC	High Efficiency Video Coding (ISO/IEC 23008-2)
AVC	Advanced Video Coding (ISO/IEC 14496-10)

V-attr

Value	Reference
8K	Resolution of up to 7680 x 4320
4K	Resolution of up to 3840 x 2160
HD	Resolution of up to 1920 x 1080
HLG	Hybrid Log-Gamma (ITU-R BT.2100)
PQ	Perceptual Quantization (ITU-R BT.2100)
SDR	Standard dynamic range with color gamut of ITU-R BT.709 or IEC 61966-2-4
SDR2020	Standard dynamic range with wide color gamut of ITU-R BT.2020
25	Frame rate of up to 25 frame/sec (e.g. 25p, 23.976p, 50i)
30	Frame rate of up to 30 frame/sec except 25 frame/sec (e.g. 30p, 29.97p, 23.976p, 60i, 59.94i)
50	Frame rate of up to 50 frame/sec, except 30 frame/sec
60	Frame rate of up to 60 frame/sec, except 50 and 25 frame/sec
100	Frame rate of up to 100 frame/sec, except 60 and 30 frame/sec
120	Frame rate of up to 120 frame/sec, except 100, 50 and 25 frame/sec

A-codec

Value	Reference
M4AAC	MPEG-4 AAC (ISO/IEC 14496-3)
M4ALS	MPEG-4 ALS (ISO/IEC 14496-3)
M2AAC	MPEG-2 AAC (ISO/IEC 13818-7)

A-attr

Value	Reference
MULT5	More than 2 and up to 5.1 channels*
MULT7	More than 5.1 and up to 7.1 channels
MULT22	More than 7.1 and up to 22.2 channels

*: When the number of channels is 1 or 2, A-attr about number of channels is omitted.

The following are examples of the apn-value:

- DTCP2_MMTTLV_HEVC+4K+HLG_M4AAC

- DTCP2_TTS_HEVC+2K+SDR2020_M4AAC

When "DTCP.COM_PN=" with an apn-value is used for content, the content needs to be compliant with all of the referred standards and conditions corresponding to elements of such apn-value. The apn-values for “DTCP.COM_PN” may only contain the values defined in this section.

The apn-values specified in the other specifications may contain undefined value(s) if no applicable value is defined in this section. However, such undefined value(s) must begin with a lower-case alphabetical character to show that the value is not defined in the DTCP2 Specification.

11 Annex

11.1 UUID of DTCP2

The Universally Unique Identifier for DTCP2[22] is defined as follows in accordance with the specification of the version 4 UUID of RFC 4122:

71f0b758-153d-4a56-998d-e113eedc97d5

[22] DTCP2 specification for other transport may define different UUID.