ICS 35.200
L 63

Social Organization Standard

T/SUCA 031—2022

Technical Specification of Advanced Digital Content Protection System

(English translation)

Issue date: 2022 – 09 – 06 Implementation date: 2022 – 12 – 06

Issued by Shenzhen 8K UHD Video Industry Cooperation Alliance

Contents

Foreword

This document is drafted in accordance with GB/T 1.1-2020 *Directives for standardization - Part 1: Rules for the structure and drafting of standardizing documents*.

Please note that certain content in this document may involve patents. The issuing organization of this document does not assume responsibility for identifying these patents.

This document was proposed and managed by the Shenzhen 8K Ultra-High Definition Video Industry Collaboration Alliance.

Drafting organizations of this standard: Shenzhen National Engineering Laboratory of Digital Television Co., Ltd., Xidian University, HiSilicon (Shanghai) Technologies Co., Ltd., SMIT Group (Shenzhen) Limited, Shenzhen 8K UHD Video Industry Cooperation Alliance, Shenzhen CESI Information Technology Co., Ltd., Shenzhen TCL New Technology Co., Ltd., Lenovo (Shenzhen) Electronics Co., Ltd., Shenzhen Skyworth-RGB Electronics Co., Ltd., ShenZhen Media Group, Shenzhen Sannuo Information Technology Co., Ltd., and ShenZhen Digital Certificate Authority Center Company Limited.

Main drafters of this standard: Li Xinguo, Gao Ming, Liang Zhijian, Liu Huayu, Yu Xiaolong, Pang Xufei, Han Gaodian, Wu Xiaoping, Lian Qiaozhen, Xu Yaoling, Huang Kunhua, Jiang Guizhu, Wang Zhihui, Zhu Zhengyuan, He Huimin, Feng Nanfei, Liang Jiyun, Le Penghui, and Zhang Zhe.

Technical Specification of Advanced Digital Content Protection System

1 Scope

This document specifies the technical requirements for secure transmission of digital content over the digital multimedia interface, including device authentication and key negotiation, rights control, content encryption and decryption, and trust system.

This document is applicable to the design, development, test, and application of the content protection system for digital multimedia interface. It can be used as a reference for transmission between other electronic devices.

2 Normative References

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

GB/T 25069-2022 Information security techniques - Terminology

GB/T 17964 Information security technology - Modes of operation for a block cipher

GB/T 20518 Information security technology - Public key infrastructure - Digital certificate format

GB/T 32905 Information security techniques - SM3 cryptographic hash algorithm

GB/T 32907 Information security techniques - SM4 block cryptographic algorithm

GB/T 32918.1 Information security technology - Public key cryptographic algorithm SM2 based on elliptic curves - Part 1: General

GB/T 32918.2 Information security technology - Public key cryptographic algorithm SM2 based on elliptic curves - Part 2: Digital signature algorithm

GB/T 32918.5 Information security technology - Public key cryptographic algorithm SM2 based on elliptic curves - Part 5: Parameter definition

GM/T 0005 Randomness test specification

T/SUCA 001.1 General purpose multimedia interface specification - Part 1: Architecture

T/SUCA 001.2 General purpose multimedia interface specification - Part 2: Protocol

RFC 2104 HMAC: keyed-hashing for message authentication

RFC 5869 HMAC-based extract-and-expand key derivation function (HKDF)

RFC 6090 fundamental elliptic curve cryptography algorithms

3 Terms and Definitions

The following terms and definitions and those defined in GB/T 25069-2022 are applicable to this document.

3. 1

AV stream transmitter

An AV stream transmitter is the start point of AV stream transmission. It packages an AV stream and sends it to the receiver.

3. 2

AV stream receiver

An AV stream receiver is the end point of AV stream transmission. It receives an AV stream and displays it.

3. 3

Authentication initiator

An authentication initiator is an AV stream transmitter that initiates device authentication to an AV stream receiver.

3. 4

Authentication responder

An authentication responder is an AV stream receiver that responds to the authentication initiated by an authentication initiator.

3. 5

Content key

A content key is used to encrypt and decrypt an AV stream.

3. 6

Device identity

A device identity is used to uniquely identify an AV stream transmitter or receiver.

3. 7

AV stream identity

An AV stream identity is used to identify an AV stream.

3. 8

Rights control policy

A rights control policy is used to control the authorization of the AV stream receiver.

3. 9

Key distribution packet

A key distribution packet carries data such as the content key.

3. 10

Encryption description packet

An encryption description packet carries data such as the encryption algorithm and the content identifier.

4 Symbols and Abbreviations

4.1 Symbols

Table 1 describes the symbols used in this document.

Table 1 Symbol conventions

Symbol	Definition
||	Data concatenation.
AlgID_A	Algorithm suite identifier used by device A, 8 bits in total. The high 4 bits indicate the authentication algorithm suite supported by the device. The binary value 0001b indicates that authentication algorithm suite 1 is supported. Other values are reserved. For details about the algorithms contained in the authentication algorithm suite, see Appendix A. The low 4 bits indicate the content encryption algorithm supported by the device. The binary value 0001b indicates that the SM4-CTR algorithm is supported. Other values are reserved.
AlgID_B	Algorithm suite identifier used by device B, 8 bits in total. The high 4 bits indicate the authentication algorithm suite supported by the device. The binary value 0001b indicates that authentication algorithm suite 1 is supported. Other values are reserved. For details about the algorithms contained in the authentication algorithm suite, see Appendix A. The low 4 bits indicate the content encryption algorithm supported by the device. The binary value 0001b indicates that the SM4-CTR algorithm is supported. Other values are reserved.
Cert_A	Certificate chain of device A, including the device certificate and the sub-CA certificate that issues the device certificate.
Cert_B	Certificate chain of device B, including the device certificate and the sub-CA certificate that issues the device certificate.
CRL_ThisUpdate_A	Issuance time of the CRL stored on device A. It is the total number of seconds from 00:00:00 on January 1, 1970 (GMT). The value is a 32-bit unsigned integer.
CRL_ThisUpdate_B	Issuance time of the CRL stored on device B. It is the total number of seconds from

	00:00:00 on January 1, 1970 (GMT). The value is a 32-bit unsigned integer.
CRLReq_HMAC	Message authentication code generated by performing HMAC calculation on the CRL request message (MCRLReq) using the specified authentication key.
CRLRsp_HMAC	Message authentication code generated by performing HMAC calculation on the CRL response message (MCRLRsp) using the specified authentication key.
CRLUpdate_HMAC	Message authentication code generated by performing HMAC calculation on the CRL update message (MCRLUpdate) using the specified authentication key.
CRLUpdateACK_HMAC	Message authentication code generated by performing HMAC calculation on the CRL update acknowledgment message (MCRLUpdateACK) using the specified authentication key.
DH_A	Temporary private parameter of device A in the Diffie–Hellman (DH) key negotiation algorithm. For details about how to generate the parameter when the SM2 algorithm is used, see section 6.1 of GB/T 32918.1.
DH_B	Temporary private parameter of device B in the DH key negotiation algorithm B. For details about how to generate the parameter when the SM2 algorithm is used, see section 6.1 of GB/T 32918.1.
DHPK_A	Temporary public parameter of device A in the DH key negotiation algorithm.
DHPK_B	Temporary public parameter of device B in the DH key negotiation algorithm.
DHPK_List_A	DHPK parameter list of device A. If device A supports multiple algorithm suites, multiple DHPKs are organized into DHPK_List_A in sequence. The current version supports only authentication algorithm suite 1. Therefore, DHPK_List_A contains only the parameters corresponding to algorithm suite 1.
DHSK	Negotiated shared key. For details about the key negotiation mechanism, see RFC 6090. The algorithm is determined based on the elliptic curve algorithm of the authentication algorithm suite. When the SM2 algorithm is used, DHSK is the x coordinate of the negotiated shared key, which is 256 bits.
HASH(msg)	Hash calculation of msg. The algorithm is determined based on the hash algorithm of the authentication algorithm suite.
HMAC(K, M)	Message authentication code calculated for M using the authentication key K. The algorithm is determined based on the message authentication algorithm of the authentication algorithm suite.
ID_A	Unique ID of device A. If device A does not have a certificate, the ID is stored after being randomly generated for the first time. If device A has a certificate, the ID is the unique device ID in the common name of the certificate subject.
ID_B	Unique ID of device B. If device B does not have a certificate, the ID is stored after being randomly generated for the first time. If device B has a certificate, the ID is the unique device ID in the common name of the certificate subject.
KDF(K, salt, info, len)	Key expansion algorithm. The HKDF scheme in RFC 5869 is used. K is the original key to be expanded, salt is the salt value, info is the key-related information, and len is the bit length of the derived key. The algorithm is determined based on the key

	derivation algorithm of the authentication algorithm suite.
KHMAC_A	Authentication key, which is used by device A to perform HMAC calculation on the protocol interaction message.
KHMAC_B	Authentication key, which is used by device B to perform HMAC calculation on the protocol interaction message.
Km	Shared master key, which is shared by both parties after authentication and key negotiation.
Msg_HASH	Hash value generated based on all messages exchanged in the protocol.
Msg_HMAC	Message authentication code generated by performing HMAC calculation on MsgHash using the specified authentication key.
PrK_A	Private key corresponding to the certificate of device A.
PrK_B	Private key corresponding to the certificate of device B.
PuK_A	Public key corresponding to the certificate of device A.
PuK_B	Public key corresponding to the certificate of device B.
S_A	Signature value generated using PrK_A.
S_B	Signature value generated using PrK_B.
Sign(PrK_X, M)	Digital signature of M using the private key PrK_*X* of device *X*. The algorithm is determined based on the signature algorithm of the authentication algorithm suite.

4.2 Abbreviations

The following abbreviations are applicable to this document.

ADCP: Advanced Digital Content Protection

AIR: Authentication Information Record

ASP: Audio Sample Packet

AVP: Active Video Packet

CK: Content Key

CKEK: Content Key Encryption Key

CRL: Certificate Revocation List

CTR: Counter

ECK: Encrypted Content Key

EDP: Encryption Description Packet

KDP: Key Distribution Packet

PKI: Public Key Infrastructure

uimsbf: unsigned integer, most significant bit first

5 System Architecture

5.1 Overview

The ADCP system authenticates devices based on the public key infrastructure (PKI) certificate system and encrypts the AV streams transmitted between devices. For details about the trust system, see chapter 9.

5.2 Logical Architecture

The ADCP system consists of the audio and video (AV) stream transmitter and receiver, as shown in Figure 1.

The AV stream transmitter is the starting point of AV stream transmission, including the authentication module, authorization module, key management module, and encryption module.

a) The authentication module implements the authentication and key negotiation for the AV stream receiver.

b) The authorization module authorizes the AV stream receiver based on the rights control policy, which is obtained from an external system.

c) In unicast scenarios (that is, the AV stream transmitter sends an AV stream to a single AV stream receiver), the key management module generates a unicast content key through key derivation. In multicast scenarios (that is, the AV stream transmitter sends an AV stream to multiple AV stream receivers), the key management module generates a multicast content key and multiple key distribution packets before sending the AV stream.

d) The encryption module generates an encryption description packet, encrypts the AV stream using the content key, adds the encryption description packet and key distribution packet (if any) to the encrypted AV stream, and sends the encrypted AV stream to the AV stream receiver through the transmission channel.

The AV stream receiver is the end point of AV stream transmission, including the authentication module, key management module, and decryption module.

a) The authentication module implements authentication and key negotiation for the AV stream transmitter.

b) In unicast scenarios, the key management module derives the content key based on the encryption description packet filtered by the decryption module. In multicast scenarios, the key management module obtains the content key distributed by the AV stream transmitter through the key distribution packets.

c) The decryption module decrypts the received AV stream using the content key.

Some routing devices may exist between the AV stream transmitter and receiver. They participate only in routing ADCP protocol messages and AV stream data, and do not involve any cryptographic operations.

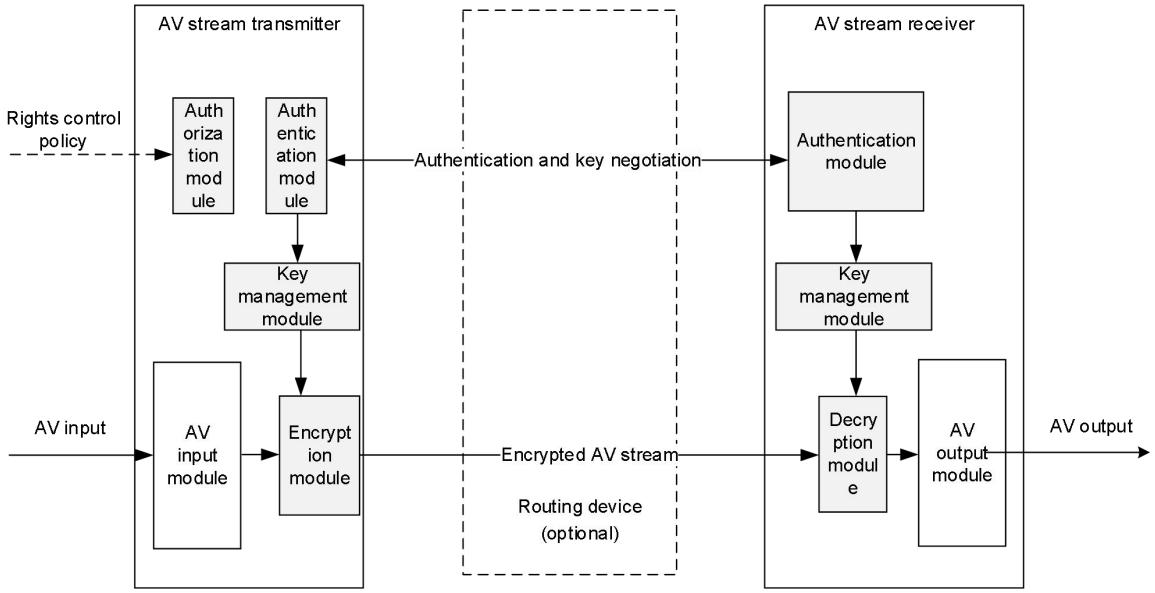

Figure 2 Logical architecture

5.3 Workflow

Figure 2 shows the working process of the ADCP system.

a) Authentication and key negotiation

Before sending an encrypted AV stream to the AV stream receiver, the AV stream transmitter needs to complete authentication and key negotiation with the AV stream receiver. The AV stream transmitter initiates an authentication request to negotiate the supported protocol version and cryptographic algorithm with the AV stream receiver, verify the certificate chain of the AV stream receiver, and negotiate the shared master key Km. For details, see chapter 6. After the authentication is completed, if both the AV stream transmitter and receiver support the CRL update protocol, they shall synchronize the CRL based on the issuance time of the locally stored CRL to ensure that the locally stored CRL is the latest version.

b) Rights control

The AV stream transmitter performs rights control on the AV stream receiver based on the rights control policy of the current AV stream. For details, see chapter 7.

c) AV stream encryption by the AV stream transmitter

1) Generation of a content key

In unicast scenarios, the AV stream transmitter generates a unicast content key through key derivation.

In multicast scenarios, the AV stream transmitter generates a multicast content key using a random number and generates key distribution packets.

2) Generation of an encryption description packet

The AV stream transmitter generates an encryption description packet.

3) Content encryption

The AV stream transmitter uses the content key to encrypt the AV stream, and packs the key distribution packet (if any) and the encryption description packet into the encrypted AV stream.

4) The AV stream transmitter sends the encrypted AV stream to the receiver.

d) AV stream decryption by the AV stream receiver

1) Filtering the key distribution packet and the encryption description packet

The AV stream receiver receives the encrypted AV stream and filters out the key distribution packet and the encryption description packet.

2) Obtaining the content key

For unicast, the AV stream receiver derives the unicast content key.

For multicast, the AV stream receiver obtains the multicast content key from the key distribution packets.

3) Content decryption

The AV stream receiver uses the content key to decrypt the AV stream.

On the GPMI, the authentication and key negotiation between the AV stream transmitter and receiver are implemented through the management adapter.

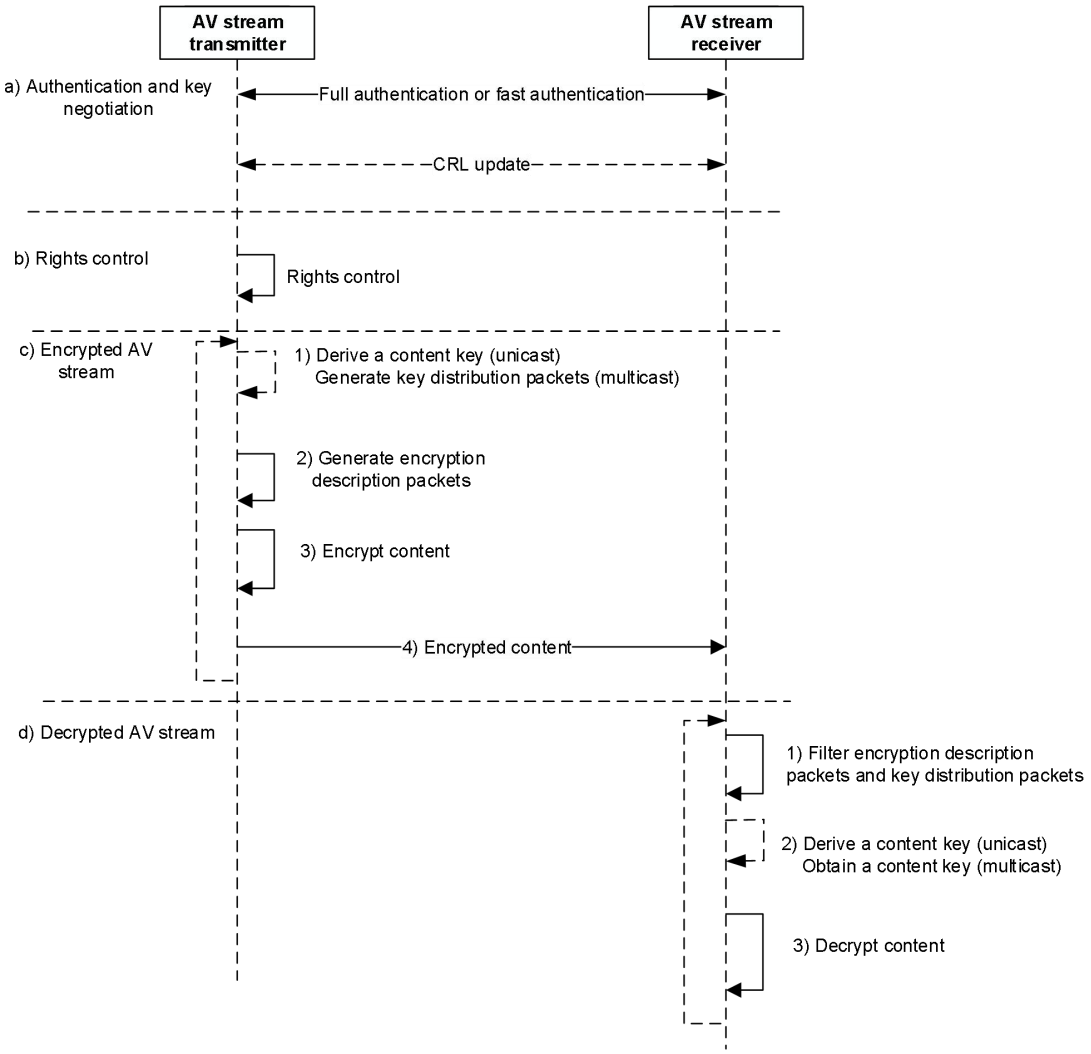

Figure 3 System working process

6 Authentication and Key Negotiation

6.1 Overview

ADCP devices perform authentication and key negotiation through the full authentication protocol or fast authentication protocol. After the authentication is successful, the certificate revocation list (CRL) is updated through the CRL update protocol, providing a unified security context for rights control based on the AV stream.

6.2 Full Authentication Protocol

The full authentication protocol is used by both parties to implement identity authentication and shared master key negotiation using the public key cryptography. Before device A sends the protected AV stream to device B, device A checks the authentication status with device B. If the authentication status is not successful, device A initiates the full authentication protocol to device B. After the authentication is completed, the two parties store or update the authentication information record (AIR). For details about the fields in the AIR, see Table 2 Authentication information record. The AIR security requirements must comply with the data security requirements in Appendix B.

Table 2 Authentication information record

Field	Meaning	Remarks
ID	Unique ID of the peer device	Can be used as the index of the AIR.
Km	Shared master key	For full authentication, it is derived from the shared key negotiated by public key parameters. For fast authentication, it is derived from the original shared master key and a random number.
FastAuth	Number of fast authentications	The value is initialized to 0 after full authentication is completed.
AlgID	Algorithm suite ID, which records the algorithm suite used for authentication and the content encryption cryptographic algorithm supported by the peer device	For details, see section 4.1.
PeerAuth	Authentication result	PeerAuth = 0 indicates that the peer device is not authenticated. PeerAuth = 1 indicates that the peer device is successfully authenticated.
Version	Version field in the authentication protocol	Used for rights control. For details, see chapter 7. This field is available only when PeerAuth = 1.
SecurityLevel	Security level field in the peer device certificate	Used for rights control. For details, see chapter 7. This field is available only when PeerAuth = 1.

SubCASerialNumber	Serial number field in the sub-CA certificate of the peer device	Used to query the CRL to check whether the certificate is revoked. This field is available only when PeerAuth = 1.
DeviceSerialNumber	Serial number field in the peer device certificate	
ProductModelID	Product model field in the peer device certificate	

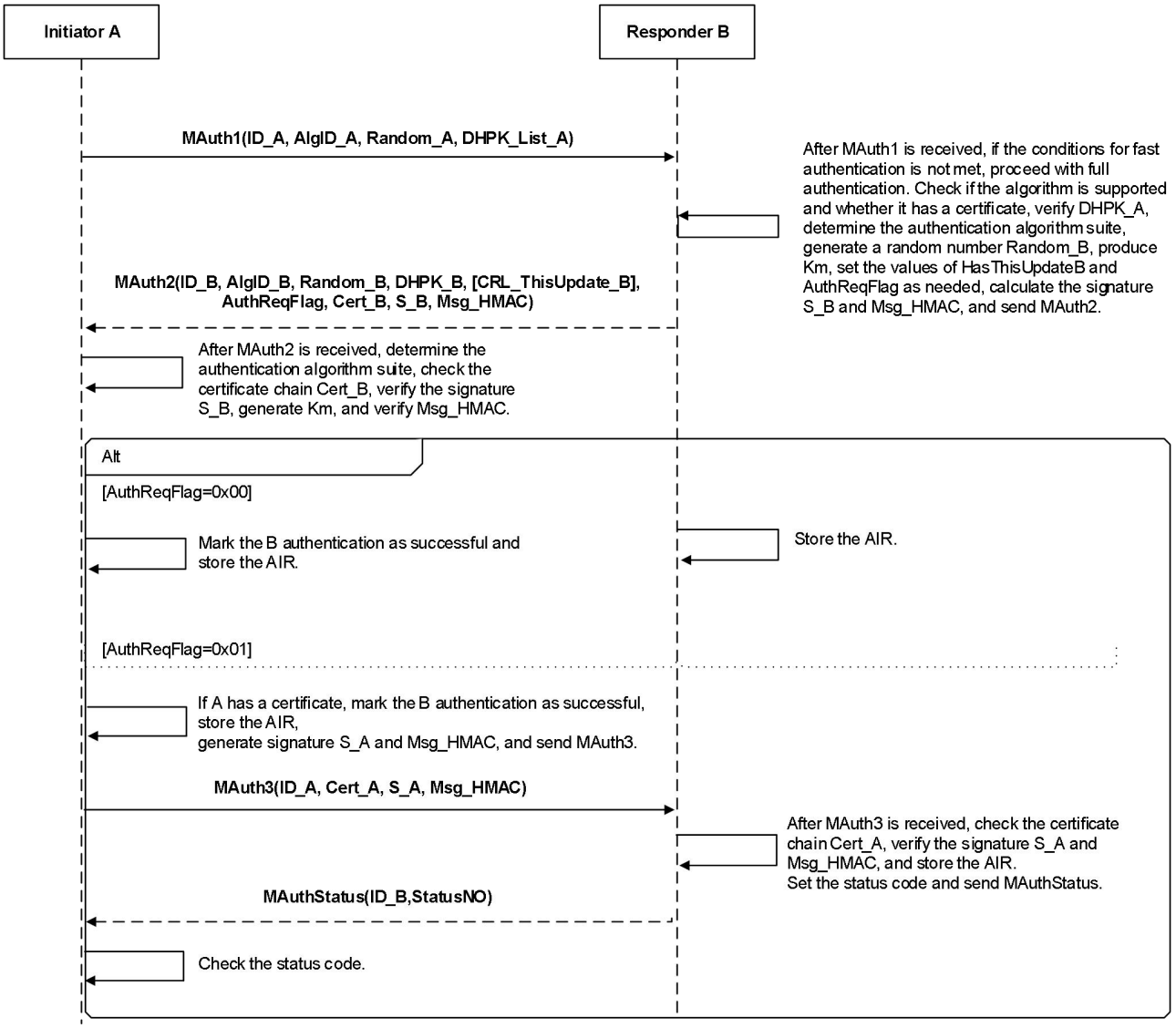

Figure 4 Full authentication process

Figure 3 shows the full authentication process. The details are as follows:

a) The authentication initiator A generates a random number Random_A. Based on the supported authentication algorithm suite and the content encryption algorithm, A sets an algorithm identifier AlgID_A, generates a private parameter DH_A, and uses DH_A to calculate a public parameter DHPK_A. If multiple authentication algorithm suites are supported, A needs to generate multiple DH_A and DHPK_A parameters and serialize them into DHPK_List_A. Then, A sends a message MAuth1 to the authentication responder B. For

details about the format of the MAuth1 message, see Table 3. When using authentication algorithm suite 1 or SM2 for negotiation, the calculation of DHPK_A must comply with GB/T 32918.1.

Table 3 MAuth1 message

Field Syntax	Number of Bits	Type	Description
MAuth1(){			
Version=0x01	8	uimsbf	Current ADCP version. The value is 0x01.
MsgID=0x11	8	uimsbf	Message ID. The value is 0x11.
Len[15..0]	16	uimsbf	Data length following this field, in bytes.
ID_A[47..0]	48	uimsbf	Unique ID of device A.
AlgID_A[7..0]	8	uimsbf	Algorithm suite ID supported by A. In the current version, only AlgID_A 0x11 is supported, indicating that authentication algorithm suite 1 is used for authentication and SM4-CTR is used for content encryption.
Random_A[127..0]	128	uimsbf	Random number generated by A.
DHPK_List_A(){			
DHPK_A_Number[7..0]	8	uimsbf	Number of DHPK_A parameters corresponding to the supported authentication algorithm suite. In the current version, the value is 1.
for(i=0;i<DHPK_A_Number;i++){			
DHPK_A_Len[7..0]	8	uimsbf	Length of DHPK_A, in bytes. In the current version, authentication algorithm suite 1 is supported, and the length is 64 bytes.
DHPK_A[DHPK_A_Len*8-1...0]	DHPK_A_Len*8	uimsbf	DH negotiation public parameter of A.
}			
}			
}			

b) The authentication responder B checks the version, message ID, and message format of the received MAuth1 message. If they are incorrect, B sets the status code to 0xF4, sends an authentication failure message MAuthStatus, and ends the authentication. For details about the MAuthStatus message format, see Table 4. Then, B checks whether an AIR corresponding to ID_A exists. If the AIR exists and FastAuth is less than 8, and PeerAuth == 1, or PeerAuth == 0 and B does not need to authenticate A, fast authentication is performed (see section 6.3). Otherwise, full authentication continues.

1) B checks the algorithm ID AlgID_A. If B does not support the algorithm that is supported by A, B sets the status code to 0xF3 and returns an authentication failure message MAuthStatus. Otherwise, B selects the authentication algorithm suite used for authentication based on the high 4 bits of AlgID_A, and proceeds to the next step. For details about the definition of the status code StatusNO of MAuthStatus, see Table 5.

   Note: During the entire authentication process, when a device receives an authentication failure message MAuthStatus, the device marks the peer authentication as failed and stops the current authentication. If the AIR has been stored, it should be deleted.

2) B obtains its device certificate DeviceCert. If B does not have a certificate, B marks the authentication with A as failed, sets the status code to 0xF5, sends an authentication failure message MAuthStatus, and ends the authentication.

3) B obtains DHPK_A of the corresponding algorithm from DHPK_list_A and verifies its validity (for details about the verification of negotiation parameters of the SM2 curve, see section 6.2.1 in GB/T 32918.1). If the verification fails, B marks the A authentication as failed, sets the status code to 0xF7, sends an authentication failure message MAuthStatus, and ends the authentication.

4) B sets the high 4 bits of AlgID_B based on the authentication algorithm suite used in this authentication. B also determines the content encryption algorithm and sets the low 4 bits of AlgID_B based on the content encryption algorithm supported by B and the low 4 bits of AlgID_A.

5) Additionally, B generates a random number Random_B, generates a DH negotiation private parameter DH_B, and calculates a public parameter DHPK_B based on the selected algorithm (the calculation method is the same as that of DHPK_A). B uses DH_B and DHPK_A to calculate the DH shared key DHSK (for details about the calculation method, see RFC 6090), and then derives the master key Km and authentication key KHMAC_B.

   Km=KDF(DHSK, Random_A||Random_B, "MainKey"||DHPK_A||DHPK_B, 256)

   KHMAC_B=KDF(Km, Random_A||Random_B, "HMACKey", 256)

6) If B supports the CRL update protocol described in this document, HasThisUpdateB is set to 1. Otherwise, HasThisUpdateB is set to 0.

7) If B needs to authenticate A, AuthReqFlag is set to 1. Otherwise, AuthReqFlag is set to 0.

8) B sends a full authentication message MAuth2 to A. For details about the message format of MAuth2, see Table 6. If AuthReqFlag in step 7 is set to 0, B sets PeerAuth to 0, generates and stores the AIR.

Table 4 MAuthStatus message

Field Syntax	Number of Bits	Type	Description
MAuthStatus(){			
Version=0x01	8	Uimsbf	Current ADCP version. The value is 0x01.
MsgID=0x15	8	Uimsbf	Message ID. The value is 0x15.
Len[15..0]	16	Uimsbf	Data length following this field, in bytes.
ID[47..0]	48	Uimsbf	Unique ID of the device that sends the message. The value is ID_B.
StatusNO[7..0]	8	Uimsbf	Status code. For details, see Table 5.
}			

Table 5 Status code definition

Field	Meaning
0x00	Normal status response.
0xF1	Status error. The version is not supported.
0xF2	Status error. The message ID is not supported.
0xF3	Status error. The algorithm is not supported.
0xF4	Status error. The message format is incorrect. For example, the message field does not match or the length is incorrect.
0xF5	Status error. The certificate does not exist.
0xF6	Status error. The certificate chain is revoked or fails to be verified.
0xF7	Status error. The DHPK verification is incorrect.
0xF8	Status error. The message verification is incorrect. For example, the signature is incorrect or the message authentication code is incorrect.

Table 6 MAuth2 message

Field Syntax	Number of Bits	Type	Description
MAuth2(){			
Version=0x01	8	uimsbf	Current ADCP version. The value is 0x01.
MsgID=0x12	8	uimsbf	Message ID. The value is 0x12.
Len[15..0]	16	uimsbf	Data length following this field, in bytes.
ID_B[47..0]	48	uimsbf	Unique ID of device B.
AlgID_B[7..0]	8	uimsbf	Algorithm suite ID selected by device B.
Random_B[127..0]	128	uimsbf	Random number generated by device B.
DHPK_B_Len[7..0]	8	uimsbf	Length of DHPK_B, in bytes.
DHPK_B[DHPK_B_Len*8-1..0]	DHPK_B_Len*8	uimsbf	DH negotiation public parameter of device B.
HasThisUpdateB[7..0]	8	uimsbf	Whether CRL_ThisUpdate_B is available.
if (HasThisUpdateB==1) {			
CRL_ThisUpdate_B[31..0]	32	uimsbf	Issuance time of the CRL stored on device B. This field is optional.
}			
AuthReqFlag[7..0]	8	uimsbf	Whether device B needs to authenticate device A.
Cert_B(){			
DeviceCert_Len[15..0]	16	uimsbf	Certificate length of device B, in bytes.
DeviceCert[DeviceCert_Len*8-1..0]	DeviceCert_Len*8	uimsbf	Device certificate of device B.
SubCACert_Len[15..0]	16	uimsbf	Sub-CA certificate length of

			device B, in bytes.
SubCACert[SubCACert_Len*8-1..0]	SubCACert_Len*8	uimsbf	Sub-CA certificate of device B.
}			
S_B_Len[7..0]	8	uimsbf	Signature data length, in bytes.
S_B[S_B_Len*8-1..0]	S_B_Len*8	Uimsbf	Signature, which is generated by using the device private key to sign MsgHash from all messages exchanged before the S_B_Len field. The private key of the current device is PrK_B.
Msg_HMAC_Len[7..0]	8	Uimsbf	Length of the message authentication code, in bytes.
Msg_HMAC[Msg_HMAC_Len*8-1..0]	Msg_HMAC_Len*8	Uimsbf	Message authentication code, which is generated by performing HMAC calculation on Msg_Hash from all messages exchanged before the S_B_Len field. The current authentication key is KHMAC_B.
}			

Note: S_B and Msg_HMAC are generated as follows:

Msg_Hash is generated from all messages exchanged in the protocol. The protocol messages are generated by concatenating all messages from MAuth1 to MAuth2 and from Version to Cert_B.

Msg_Hash=HASH(MAuth1||[MFastAuth2]||[MFastAuthToFullAuth]

||MAuth2.Version||..||MAuth2.Cert_B)

In the preceding formula, MFastAuth2 and MFastAuthToFullAuth (if available) are defined in section 6.3. S_B is generated by signing Msg_Hash using the device private key PrK_B.

S_B=Sign(PrK_B, Msg_Hash)

Msg_HMAC is calculated using the authentication key KHMAC_B.

Msg_HMAC=HMAC(KHMAC_B, Msg_Hash)

c) A checks the version, message ID, and message format of the received MAuth2 message. If they are incorrect, A sets the status code to 0xF4, sends an authentication failure message MAuthStatus, and ends the authentication. Otherwise, A performs the following steps:

1) Check the received AlgID_B. If A does not support the algorithm selected by B, A marks the B authentication as failed, sets the status code to 0xF3, sends an authentication failure message MAuthStatus to B, and ends the authentication.

2) Determine the algorithm suite used for the current authentication based on the high 4 bits of AlgID_B and verify the validity of DHPK_B of the corresponding algorithm. The verification method is the same as that in step 3 in step b). If the verification fails, A marks the B authentication as failed, sets the status code to 0xF7, sends an authentication failure message MAuthStatus to B, and ends the authentication.

3) Check the certificate chain Cert_B of B. If the certificate chain is revoked or fails to be verified, A marks the B authentication as failed, sets the status code to 0xF6, sends an authentication failure message MAuthStatus to B, and ends the authentication.

4) Use the public key PuK_B in the certificate of device B to verify the signature S_B. If the verification fails, A marks the B authentication as failed, sets the status code to 0xF8, sends an authentication failure message MAuthStatus to B, and ends the authentication.

5) Use DH_A and DHPK_B to calculate the DH shared key DHSK, and derive the master key Km and authentication key KHMAC_A:

Km=KDF (DHSK, Random_A||Random_B, "MainKey"||DHPK_A||DHPK_B, 256).

KHMAC_A=KDF(Km, Random_A||Random_B, "HMACKey", 256)

Use KHMAC_A to calculate and verify the message authentication code Msg_HMAC. If the verification fails, A marks the B authentication as failed, sets the status code to 0xF8, sends an authentication failure message MAuthStatus to B, and ends the authentication.

6) If AuthReqFlag == 0, or if AuthReqFlag == 1 and A has a certificate, A marks the B authentication as successful, sets PeerAuth to 1, generates and stores the AIR. If AuthReqFlag == 0, the full authentication process ends. If AuthReqFlag == 1 and A has a certificate, A sends a message MAuth3 to B. For details about the message format of MAuth3, see Table 7. If AuthReqFlag == 1 but A does not have a certificate, A marks the B authentication as failed, sets the status code to 0xF5, sends an authentication failure message MAuthStatus to B, and ends the authentication.

Table 7 MAuth3 message

Field Syntax	Number of Bits	Type	Description
MAuth3(){			
Version=0x01	8	Uimsbf	Current ADCP version. The value is 0x01.
MsgID=0x13	8	Uimsbf	Message ID. The value is 0x13.
Len[15..0]	16	Uimsbf	Data length following this field, in bytes.
ID_A[47..0]	48	Uimsbf	Unique ID of device A.
Cert_A(){			
DeviceCert_Len[15..0]	16	uimsbf	Certificate length of device A, in bytes.
DeviceCert[DeviceCert_Len*8-1..0]	DeviceCert_Len*8		Certificate of device A.
SubCACert_Len[15..0]	16	uimsbf	Sub-CA certificate length of device A, in bytes.
SubCACert[SubCACert_Lene*8-1..0]	SubCACert_Len*8		Sub-CA certificate of device A.
}			
S_A_Len[7..0]	8	uimsbf	Signature data length, in bytes.
S_A[S_A_Len*8-1..0]	S_A_Len*8	uimsbf	Signature, which is generated by using the device private key to sign MsgHash from all messages exchanged before the S_A_Len

			field. The private key of the current device is PrK_A.
Msg_HMAC_Len[7..0]	8	uimsbf	Length of the message authentication code, in bytes.
Msg_HMAC[Msg_HMAC_Len*8-1..0]	Msg_HMAC_Len*8	uimsbf	Message authentication code, which is generated by performing HMAC calculation on MsgHash from all messages exchanged before the S_A_Len field. The current authentication key is KHMAC_A.
}			

Note: S_A and Msg_HMAC are generated as follows:

Msg_Hash is generated from all messages exchanged in the protocol. The protocol messages are generated by concatenating all messages from MAuth1 to MAuth3 and from Version to Cert_A.

Msg_Hash=HASH(MAuth1||[MFastAuth2]||[MFastAuthToFullAuth]

||MAuth2||MAuth3.Version||..||MAuth3.Cert_A)

In the preceding formula, MFastAuth2 and MFastAuthToFullAuth (if available) are defined in section 6.3.

S_A is generated by signing Msg_Hash using the device private key PrK_A.

S_A=Sign(PrK_A, Msg_Hash)

Msg_HMAC is calculated using the authentication key KHMAC_A.

Msg_HMAC=HMAC(KHMAC_A, Msg_Hash)

d) B checks the version, message ID, and message format of the received MAuth3 message. If the check fails, B sets the status code to 0xF4, sends an authentication failure message MAuthStatus, and ends the authentication. If the check succeeds, B performs the following steps:

   1) Check the certificate chain Cert_A of A. If the certificate chain is revoked or fails to be verified, B marks the A authentication as failed, sets the status code to 0xF6, sends an authentication failure message MAuthStatus to A, and ends the authentication.
   2) Use the public key PuK_A in the certificate of device A to verify the signature S_A in MAuth3. If the verification fails, B marks the A authentication as failed, sets the status code to 0xF8, sends an authentication failure message MAuthStatus to A, and ends the authentication.
   3) Use KHMAC_B to calculate and verify the message authentication code Msg_HMAC. If the verification is successful, set PeerAuth to 1, generate and store the AIR, and set the status code to 0x00. B sends the MAuthStatus message to A to complete the authentication. If the verification fails, B marks the A authentication as failed, sets the status code to 0xF8, sends the MAuthStatus message to A, and ends the authentication.

e) After receiving the MAuthStatus message, A checks the status code. If the status code is 0x00, the authentication is completed. If the status code is not 0x00, A clears the AIR information of B, marks the B authentication as failed, and ends the authentication.

After A authenticates B based on the MAuth2 message successfully, A can start rights control on the AV stream receiver and send the encrypted AV stream to B. After B authenticates A based on the MAuth3 message successfully, B can start rights control on A and send the encrypted AV stream to A. This helps ensure better performance.

6.3 Fast Authentication Protocol

Fast authentication is performed when the initiator A and the responder B store the AIR recorded after the last authentication and have negotiated a shared master key. The AIR information is used for authentication to obtain a fresh shared master key.

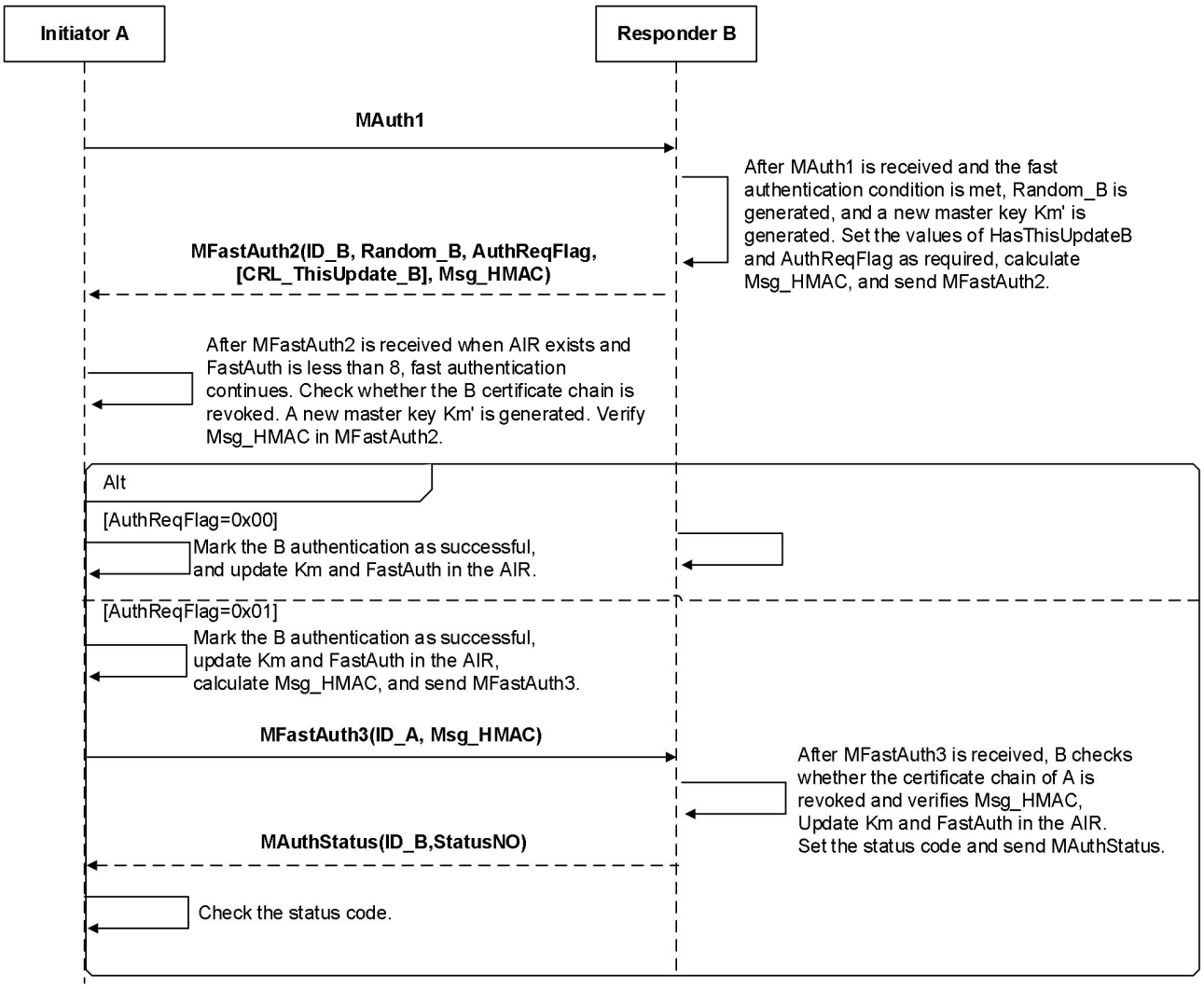

Figure 5 Fast authentication process

B checks the version, message ID, and message format of the received MAuth1 message. If they are incorrect, B sets the status code to 0xF4 and sends an authentication failure message MAuthStatus. Then, B determines whether the current authentication satisfies the fast authentication condition (see step b) in section 6.2 "Full Authentication Protocol"). If yes, the fast authentication protocol is used, as shown in Figure 4. The details are as follows:

a) B generates a random number Random_B and derives a new master key Km' and authentication key KHMAC_B from the master key Km of the stored AIR.

Km'=KDF(Km, Random_A||Random_B, "MainKey", 256)
KHMAC_B=KDF(Km', Random_A||Random_B, "HMACKey", 256)

If B supports the CRL update protocol described in this document, HasThisUpdateB is set to 1; otherwise, HasThisUpdateB is set to 0. If B needs to authenticate A, AuthReqFlag is set to 1; otherwise, AuthReqFlag is set to 0. B sends the fast authentication protocol message MFastAuth2 to A. For details about the message format of MFastAuth2, see Table 8. When AuthReqFlag is set to 0, B sets FastAuth to FastAuth + 1 and uses Km' to update Km in the AIR.

Table 8 MFastAuth2 message

Field Syntax	Number of Bits	Type	Description
MFastAuth2(){			
Version=0x01	8	uimsbf	Current ADCP version. The value is 0x01.
MsgID=0x16	8	uimsbf	Message ID. The value is 0x16.
Len[15..0]	16	uimsbf	Data length following this field, in bytes.
ID_B[47..0]	48	uimsbf	Unique ID of device B
Random_B[127..0]	128	uimsbf	Random number generated by device B.
HasThisUpdateB[7..0]	8	uimsbf	Whether to send CRL_ThisUpdate_B.
If(HasThisUpdateB==0x01){			
CRL_ThisUpdate_B[31..0]	32	uimsbf	Issuance time of the CRL stored on device B.
}			
AuthReqFlag[7..0]	8	uimsbf	Whether device B needs to authenticate device A.
Msg_HMAC_Len[7..0]	8	uimsbf	Length of the message authentication code, in bytes.
Msg_HMAC[Msg_HMAC_Len*8-1..0]	Msg_HMAC_Len*8	uimsbf	Message authentication code, which is generated by performing HMAC calculation on MsgHash from all messages exchanged before the Msg_HMAC_Len field. The current authentication key is KHMAC_B.
}			

Note: Msg_HMAC is generated as follows:

Msg_Hash is generated from all messages exchanged in the protocol. The protocol messages are generated by concatenating all messages from MAuth1 to MFastAuth2 and from Version to AuthReqFlag.

Msg_Hash=HASH(MAuth1||

||MFastAuth2.Version||..|| MFastAuth2.AuthReqFlag)

Msg_HMAC is calculated using the authentication key KHMAC_B.

$$Msg\_HMAC=HMAC(KHMAC\_B, Msg\_Hash)$$

b) A checks the version, message ID, and message format of the received MFastAuth2 message. If they are incorrect, B sets the status code to 0xF4 and sends an authentication failure message MAuthStatus. Then, A queries the AIR corresponding to ID_B. If A does not find the AIR or FastAuth in the AIR is greater than or equal to 8, A clears the AIR information corresponding to B and sends an MFastAuthToFullAuth message to B. For details about the MFastAuthToFullAuth message format, see Table 9.

If the AIR corresponding to device B exists and FastAuth is less than 8, device A continues the fast authentication and performs the following steps:

1) A extracts SubCASerialNumber, DeviceSerialNumber, and productModelID from the stored AIR, and checks whether the certificate chain of B is revoked. If yes, A marks the B authentication as failed, sets the status code to 0xF6, sends an authentication failure message MAuthStatus to device B, clears the AIR corresponding to B, and ends the authentication.
2) A derives a new master key Km' and authentication key KHMAC_A using the master key Km stored in the AIR. The method is the same as that in step a).

$$Km'=KDF(Km, Random\_A||Random\_B, "MainKey", 256)$$

$$KHMAC\_A=KDF(Km', Random\_A||Random\_B, "HMACKey", 256)$$

3) A calculates and verifies a message authentication code Msg_HMAC of MFastAuth2 by using KHMAC_A. If the verification succeeds, A marks the B authentication successful, sets FastAuth to FastAuth + 1, and updates Km in the AIR by using Km'. If the Msg_HMAC verification fails, A sets the status code to 0xF8, sends an authentication failure message MAuthStatus to B, marks the B authentication as failed, clears the AIR corresponding to B, and ends the authentication.
4) If AuthReqFlag == 1, A sends a fast authentication protocol message MFastAuth3 to B. The syntax format of the MFastAuth3 message is described in Table 10.

Table 9 MFastAuthToFullAuth message

Field Syntax	Number of Bits	Type	Description
MFastAuthToFullAuth(){			
Version=0x01	8	uimsbf	Current ADCP version. The value is 0x01.
MsgID=0x17	8	uimsbf	Message ID. The value is 0x17.
Len[15..0]	16	uimsbf	Data length following this field, in bytes.
ID_A[47..0]	48	uimsbf	Unique ID of device A.
}			

Table 10 MFastAuth3 message syntax

Field Syntax	Number of Bits	Type	Description
MFastAuth3(){			
Version=0x01	8	uimsbf	Current ADCP version. The value is 0x01.

MsgID=0x18	8	uimsbf	Message ID. The value is 0x18.
Len[15..0]	16	uimsbf	Data length following this field, in bytes.
ID_A[47..0]	48	uimsbf	Unique ID of device A.
Msg_HMAC_Len[7..0]	8	uimsbf	Length of the message authentication code, in bytes.
Msg_HMAC[Msg_HMAC_Len-1..0]	Msg_HMAC_Len*8	uimsbf	Message authentication code, which is generated by performing HMAC calculation on MsgHash from all messages exchanged before the Msg_HMAC_Len field. The current authentication key is KHMAC_A.
}			

Note: Msg_HMAC is generated as follows:

Msg_Hash is generated from all messages exchanged in the protocol. The protocol messages are generated by concatenating all messages from MAuth1 to MFastAuth3 and from Version to ID_A.

Msg_Hash=HASH(MAuth1||MFastAuth2

||MFastAuth3.Version||..||MFastAuth3.ID_A)

Msg_HMAC is calculated using the authentication key KHMAC_A.

Msg_HMAC=HMAC(KHMAC_A, Msg_Hash)

c) If B receives the MFastAuthToFullAuth message, B clears the AIR corresponding to A, performs full authentication, and proceeds to 1) in step b of 6.2.

If B receives the MFastAuth3 message, B checks the version, message ID, and message format of the message. If the check fails, B sets the status code to 0xF4, sends an authentication failure message MAuthStatus, clears the AIR corresponding to A, marks the A authentication as failed, and ends the authentication. If the check succeeds, B performs the following steps:

1) Use KHMAC_B to calculate and verify the message authentication code Msg_HMAC of MFastAuth3. If the verification fails, B marks the A authentication as failed, sets the status code to 0xF8, sends an authentication failure message MAuthStatus to A, clears the AIR information corresponding to A, and ends the authentication.

2) Extract the SubCASerialNumber, DeviceSerialNumber, and ProductModelID from the AIR as required, and check whether the certificate chain of device A is revoked. If A is not revoked, B sets FastAuth to FastAuth + 1, updates Km in the AIR using Km', sets the status code to 0x00, and sends the MAuthStatus message to A, and ends the fast authentication protocol process. If A is revoked, B marks the A authentication as failed, sets the status code to 0xF6, sends an authentication failure message MAuthStatus to A, clears the AIR information corresponding to A, and ends the authentication.

d) After receiving the MAuthStatus message, A checks the status code. If the status code is 0x00, the authentication is completed. If the status code is not 0x00, A clears the AIR information of B, marks the B authentication as failed, and ends the authentication.

After A authenticates B based on the MFastAuth2 message successfully, A can start rights control on the AV stream receiver and send the encrypted AV stream to B. After B authenticates A based on the MFastAuth3 message successfully, B can start rights control on A and send the encrypted AV stream to A. This helps ensure better performance.

6.4 CRL Update

6.4.1 CRL Update Mechanism

The authentication initiator A shall support CRL update, which can be implemented using the CRL update protocol defined in this chapter or external system interfaces. For details about the external system interfaces, see Appendix C.1.

6.4.2 CRL Update Protocol

6.4.2.1 CRL Update Protocol

After the authentication is successful, if the responder B sends CRL_ThisUpdate_B in the MAuth2 or MFastAuth2 message, the initiator A needs to compare the received CRL_ThisUpdate_B with the locally maintained CRL_ThisUpdate_A. If they are different, the initiator A initiates the CRL update protocol. If the locally maintained CRL_ThisUpdate_A is later than the received CRL_ThisUpdate_B, the initiator A sends the locally maintained CRL to the responder B to update the CRL of the responder B. For details, see section 6.4.2.2. Otherwise, the initiator A requests the locally maintained CRL of the responder B to update its own CRL. For details, see section 6.4.2.3.

6.4.2.2 CRL Update

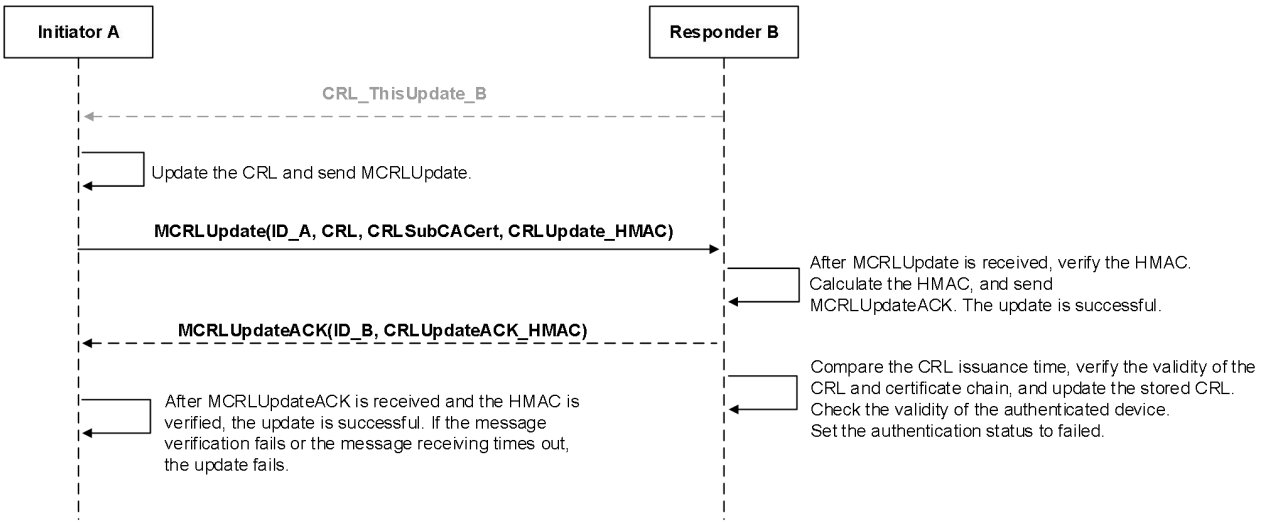

Figure 6 CRL update process

a) If CRL_ThisUpdate_B is earlier than the locally maintained CRL_ThisUpdate_A, A uses the shared master key Km negotiated with B to derive the authentication key KHMAC_CRL.

KHMAC_CRL=KDF(Km, Random_A||Random_B, "HMACCRLKey", 256)

A generates a MCRLUpdate message and sends it to B. Table 11 describes the format of the MCRLUpdate message.

Table 11 MCRLUpdate message

Field Syntax	Number of Bits	Type	Description
MCRLUpdate(){			
CRLUpdate_Info(){			
Version=0x01	8	Uimsbf	Current ADCP version. The value is

			0x01.
MsgID=0x20	8	Uimsbf	Message ID. The value is 0x20.
Len[15..0]	16	Uimsbf	Data length following this field, in bytes.
ID_A[47...0]	48	Uimsbf	Unique device ID
CRL_Length[23...0]	24	Uimsbf	CRL length, in bytes
CRL[CRL_Length*8-1...0]	CRL_Length*8	Uimsbf	Certificate revocation list
CRLSubCACert_Length[15...0]	16	Uimsbf	Length of the CRL sub-CA certificate, in bytes.
CRLSubCACert[CRLSubCACert_Length*8-1...0]	CRLSubCACert_ Length*8	Uimsbf	CRL sub-CA certificate.
}			
CRLUpdate_HMAC_Len[7..0]	8	uimsbf	Length of the message authentication code, in bytes.
CRLUpdate_HMAC[CRLUpdate_HMAC_Len*8-1...0]	CRLUpdate_HMAC_ Len*8	uimsbf	Message authentication code of CRLUpdate_Info. The current authentication key is KHMAC_CRL.
}			

b) After receiving the MCRLUpdate message, B performs the following steps:

1) Derive the authentication key KHMAC_CRL, and use KHMAC_CRL to calculate and verify the message authentication code CRLUpdate_HMAC. If the verification fails, B sets the status code to 0xF8, sends an authentication failure message MAuthStatus, and ends the CRL update process.
2) Generate a MCRLUpdateACK message and send it to A. Table 12 describes the syntax format of the MCRLUpdateACK message.
3) If the issuance time of the CRL received by B is later than that of the locally maintained CRL, B updates its CRL after verifying the validity of the CRL. The verification of CRL validity includes CRL signature verification and certificate chain verification.
4) If the CRL of B is updated, B checks whether all authenticated peers are revoked. If a peer is revoked, B marks the authentication of the peer as failed and clears the AIR information of the peer.

Table 12 MCRLUpdateACK message syntax

Field Syntax	Number of Bits	Type	Description
MCRLUpdateACK(){			
CRLUpdateACK_Info(){			
Version=0x01	8	uimsbf	Current ADCP version. The value is 0x01.
MsgID=0x21	8	uimsbf	Message ID. The value is 0x21.
Len[15..0]	16	uimsbf	Data length following this

			field, in bytes.
ID_B[47..0]	48	uimsbf	Unique ID of device B.
}			
CRLUpdateACK_HMAC_Len[7..0]	8	uimsbf	Length of the message authentication code, in bytes.
CRLUpdateACK_HMAC[CRLUpdateACK_HMAC_Len*8-1..0]	CRLUpdateACK_HMAC_Len*8	uimsbf	Message authentication code of CRLUpdateACK_Info. The current authentication key is KHMAC_CRL.
}			

c) If A receives the MCRLUpdateACK message, A uses KHMAC_CRL to calculate and verify the message authentication code CRLUpdateACK_HMAC. If the verification is successful, the CRL is updated successfully. If the verification fails, the CRL fails to be updated.

6.4.2.3 CRL Request

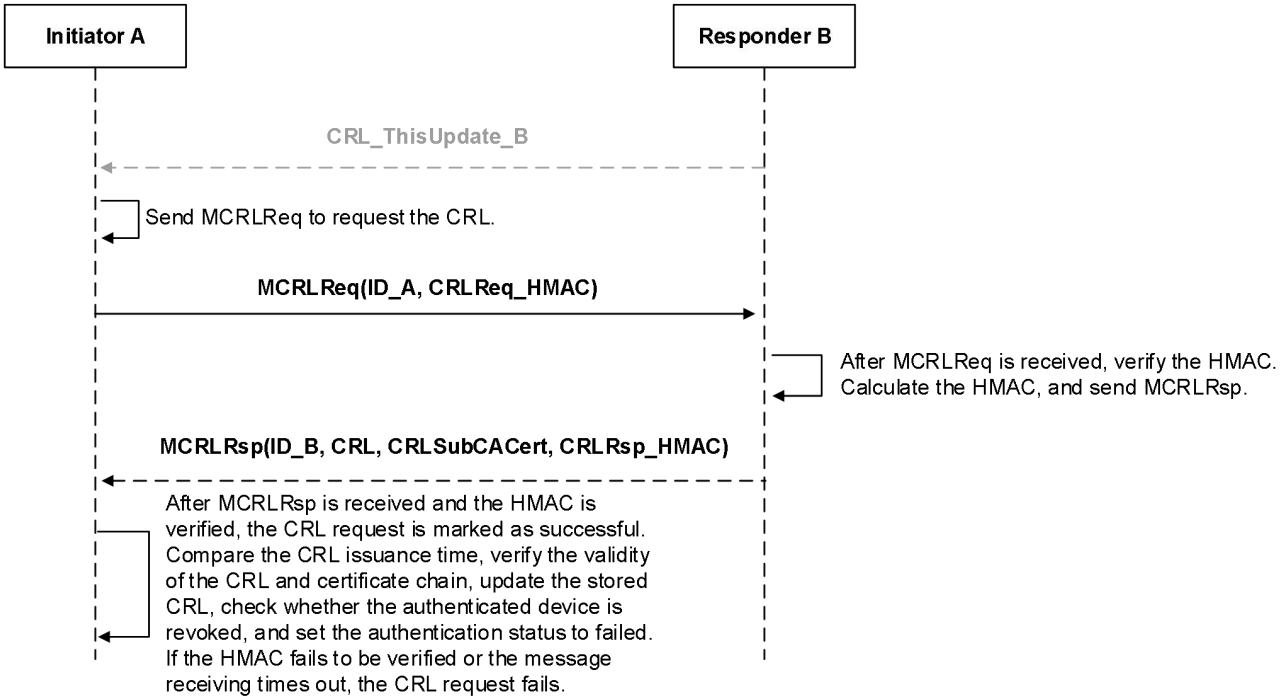

Figure 7 CRL request process

a) If CRL_ThisUpdate_B is later than the locally maintained CRL_ThisUpdate_A, A uses the shared master key Km negotiated with B to derive the authentication key KHMAC_CRL. The derivation method is the same as that in section 6.4.2.

A generates a MCRLReq message and sends it to B. For details about the MCRLReq message format, see Table 13.

Table 13 MCRLReq message format

Field Syntax	Number of Bits	Type	Description

MCRLReq(){			
CRLReq_Info(){			
Version=0x01	8	uimsbf	Current ADCP version. The value is 0x01.
MsgID=0x22	8	uimsbf	Message ID. The value is 0x22.
Len[15..0]	16	uimsbf	Data length following this field, in bytes.
ID_A[47..0]	48	uimsbf	Unique device ID
}			
CRLReq_HMAC_Len[7..0]	8	uimsbf	Length of the message authentication code, in bytes.
CRLReq_HMAC[CRLReq_HMAC_Len*8-1..0]	CRLReq_HMAC_Len*8	uimsbf	Message authentication code of CRLReq_Info. The current authentication key is KHMAC_CRL.
}			

b) After receiving the MCRLReq message, B derives the authentication key KHMAC_CRL, calculates and verifies the message authentication code CRLReq_HMAC using KHMAC_CRL. If the verification is successful, B generates a MCRLRsp message and sends it to A. For details about the MCRLRsp message format, see Table 14. If the verification fails, B sets the status code to 0xF8, sends an authentication failure message MAuthStatus, and ends the CRL request process.

Table 14 MCRLRsp message format

Field Syntax	Number of Bits	Type	Description
MCRLRsp(){			
CRLRsp_Info(){			
Version=0x01	8	uimsbf	Current ADCP version. The value is 0x01.
MsgID=0x23	8	uimsbf	Message ID. The value is 0x23.
Len[15..0]	16	uimsbf	Data length following this field, in bytes.
ID_B[47..0]	48	uimsbf	Unique device ID
CRL_Length[23..0]	24	uimsbf	CRL length, in bytes
CRL[CRL_Length*8-1..0]	CRL_Length*8	uimsbf	Certificate revocation list
CRLSubCACert_Length[15..0]	16	uimsbf	Length of the CRL sub-CA certificate, in bytes.
CRLSubCACert[CRLSubCACert_Length*8-1..0]	CRLSubCACert_Length*8	uimsbf	CRL sub-CA certificate.

}			
CRLRsp_HMAC_Len[7..0]	8	uimsbf	Length of the message authentication code, in bytes.
CRLRsp_HMAC[CRLRsp_HMAC_Len*8-1..0]	CRLRsp_HMAC_Len*8	uimsbf	Message authentication code of CRLRsp_Info. The current authentication key is KHMAC_CRL.
}			

c) After receiving the MCRLRsp message, A performs the following steps:

1) Calculate and verify the message authentication code CRLRsp_HMAC using KHMAC_CRL. If the verification is successful, the CRL request is successful.
2) If the issuance time of the CRL obtained by A is later than that of the locally maintained CRL, A updates it's CRL after verifying the validity of the CRL. The verification of CRL validity includes CRL signature verification and certificate chain verification.
3) If the CRL of A is updated, B checks whether all authenticated peers are revoked. If a peer is revoked, B marks the authentication of the peer as failed and clears the AIR information of the peer.

If A fails to verify CRLRsp_HMAC, the CRL request fails and the process ends.

If the CRL request fails, the authentication initiator sets the responder's authentication status to failed and clears the corresponding AIR.

6.5 Timeout Mechanism

During authentication and key negotiation, the authentication initiator may fail to receive a response message within 500 ms.

a) If the full authentication protocol or fast authentication protocol is in use, the authentication initiator can resend the MAuth1 message to start a new authentication process.

b) If the CRL update protocol is in use, the authentication initiator can send the CRL update protocol request. If the response message times out, the authentication initiator can re-execute the CRL update protocol up to three times. If the response still times out, the two parties fail to execute the CRL update protocol. In this case, the authentication initiator sets the responder's authentication status to failed and clears the corresponding AIR.

7 Rights Control

7.1 Rights Control Policy

Before sending an AV stream to the AV stream receiver or when the AV stream right changes, the AV stream transmitter shall authorize the AV stream receiver according to the rights control policy (RCP).

The RCP can be provided by an external system, which is not within the scope of this document. For details about the involved external system interfaces, see Appendix C.2.

Table 15 describes the rights control policy.

Table 15 Rights control policy

Field	Description
Version	Protocol version
SecurityLevel	Security level. L1 is 0x1, L2 is 0x2, L3 is 0x3, and 0x0 indicates no restriction.
Count	Maximum number of AV stream receivers. The value is fixed at 32.

Version: protocol version supported by the AV stream transmitter and receiver. The current version is 0x01. The AV stream transmitter compares the protocol version with that negotiated with the AV stream receiver during device authentication. If the negotiated protocol version is later than or equal to the protocol version, the authorization is successful. Otherwise, the authorization fails.

SecurityLevel: security level supported by the AV stream receiver. The AV stream transmitter compares the security level with that of the AV stream receiver. If the security level is lower than or equal to that of the AV stream receiver, the authorization is successful. Otherwise, the authorization fails. For details about the security level, see Appendix D. The AV stream transmitter obtains the security level from the device certificate of the AV stream receiver. For details, see chapter 9.

Count: maximum number of AV stream receivers. The value is fixed at 32. The AV stream transmitter collects statistics on the number of receivers of the current AV stream and compares the number with the count value. If the number is less than or equal to the value, the authorization is successful. Otherwise, the authorization fails.

7.2 Device Identity Collection and Count Control

The AV stream transmitter can generate a device identity list for AV stream receivers, which is accessible to external systems. In addition, the number of AV stream receivers is restricted.

Device identity collection: The AV stream transmitter obtains SubCASerialNumber, DeviceSerialNumber, and ProductModelID from the AIR of each AV stream receiver based on the receiver list. The information is summarized into a device identity list for AV stream receivers, which can be obtained by external systems. For details about the involved external system interfaces, see Appendix C.3.

Count control: The AV stream transmitter calculates the number of AV stream receivers based on the current receiver list and limit the number of AV stream receivers.

7.3 Dynamic Authorization Update or Cancellation

If the AV stream transmitter receives a new rights control policy for the current AV stream during transmission, it checks whether the existing AV stream receivers meet the new rights control policy. Receivers that meet the policy are successfully authorized, while those that do not fail authorization. Notably, even if some AV stream receivers fail authorization, those that are successfully authorized can continue to receive the AV stream.

8 Content Encryption and Decryption

8.1 Overview

Content encryption and decryption support both unicast and multicast scenarios. In unicast scenarios, both parties generate a unicast content key through key derivation. In multicast scenarios, the AV stream transmitter generates a multicast content key using a random number

and distributes the key to the AV stream receivers through key distribution packets. For details, see Appendix E.

The AV stream transmitter generates a counter for encryption purposes, creates an encryption description packet, encrypts the AV stream using the content key and the encryption counter, and packs the key distribution packet (if any) and encryption description packet into the AV stream before sending it to the AV stream receiver.

The AV stream receiver filters out the key distribution packet and the encryption description packet from the AV stream. It then uses the derived unicast content key or the multicast content key distributed via the key distribution packet to obtain the counter used for AV stream encryption, and proceeds to decrypt the AV stream.

The content key needs to be updated when:

a) An AV stream is encrypted using a content key for more than 24 hours or more than 2,592,000 frames.

b) The unicast scenario is switched to the multicast scenario.

c) The rights of an AV stream are changed in the multicast scenario.

d) The receiver device exits the receiving of an AV stream in the multicast scenario.

8.2 Unicast Content Key Derivation Mechanism

In unicast scenarios, the AV stream transmitter A and receiver B derive the content key using the shared master key Km.

CK=KDF(Km, Random_A||Random_B||ID_A||ID_B||CKId', "Unicast Content Key", 128)

To be specific:

a) Km is the shared master key negotiated by both parties.

b) Random_A and Random_B are random numbers exchanged during authentication and key negotiation.

c) ID_A is the unique ID of AV stream transmitter A, consisting of 48 bits.

d) ID_B is the unique ID of AV stream receiver B, consisting of 48 bits.

e) CKId is the identifier for the content key of an AV stream, whose value is CurCKId or NextCKId from the encryption description packet. The default value of the initial CKId for an AV stream is 0. CKId' refers to CKId with the high 2 bits padded with 0s. It is 16 bits in length. When the unicast content key requires an update, the new content key is derived by changing CKId'.

8.3 Multicast Content Key Distribution Mechanism

In multicast scenarios, the AV stream transmitter sends a content key to each receiver through key distribution packets (KDPs), which are described in Table 16.

Table 16 Key distribution packet

Field	Number of Bits	Type	Field Sequence	Description
Type	8	uimsbf	KDP0[7:0]	Data packet type. The value is 0x01.
Version	8	uimsbf	KDP1[7:0]	Version number of the KDP. The value is 0x01.
Len	8	uimsbf	KDP2[7:0]	Data length following this field, in bytes.
CKId	14	uimsbf	KDP3[7:0] KDP4[7:2]	ID of the content key in the current packet. The high 8 bits of CKId should be in the fourth byte of the KDP (KDP3), and the low 6 bits of CKId should be in the high 6 bits of the fifth byte of the KDP (KDP4).
Reserved	2	uimsbf	KDP4[1:0]	Reserved
ID_B	48	uimsbf	KDP5[7:0]... KDP10[7:0]	Unique ID of the AV stream receiver B, which uses binary coding. The first byte of ID_B should be in the sixth byte of the KDP (KDP5), and the 2nd to 6th bytes of ID_B should be arranged in KDP6 to KDP10.
ECKCtr	128	uimsbf	KDP11[7:0]... KDP26[7:0]	Counter used for encrypting the content key. The first byte of ECKCtr should be in the 12th byte of the KDP (KDP11), and the 2nd to 16th bytes of ECKCtr should be arranged in KDP12 to KDP26.
ECK	128	uimsbf	KDP27[7:0]... KDP42[7:0]	Encrypted content key. The content key encryption key CKEK and counter ECKCtr are used to encrypt the content key. CKEK is derived from the shared master key Km. For details about the encryption algorithm, see the content key encryption algorithm in Appendix A. The CKEK generation method is as follows: CKEK=KDF(Km, Random_A||Random_B||ID_A||ID_B, "Content Key Encryption Key", 128) The first byte of ECK should be in the 28th byte of the KDP (KDP27), and the 2nd to 16th bytes of ECK should be arranged in KDP28 to KDP42.
Reserved	8	uimsbf	KDP43[7:0]	Reserved

8.4 Content Encryption and Decryption Mechanism

In both unicast and multicast scenarios, the AV stream transmitter A sends an encryption description packet (EDP) to describe the content encryption and decryption mode. Table 17 describes the EDP format.

Table 17 Encryption description packet

Field	**Number of Bits**	**Type**	**Field Sequence**	**Description**
Type	8	uimsbf	EDP0[7:0]	Data packet type. The value is 0x02.
Version	8	uimsbf	EDP1[7:0]	Version number of the EDP. The value is 0x01.
Len	8	uimsbf	EDP2[7:0]	Data length following this field, in bytes.
CurCKId	14	uimsbf	EDP3[7:0] EDP4[7:2]	ID of the current content key. The high 8 bits of CurCKId should be in the fourth byte of the EDP (EDP3), and the low 6 bits of CurCKId should be in the high 6 bits of the fifth byte of the EDP (EDP4).

CurCKType	2	uimsbf	EDP4[1:0]	Type of the current content key, which is defined as follows: 00b: unicast. The content key is generated using derivation. 01b: multicast. The content key is distributed using a KDP. Others: reserved. CurCKType should be in the low 2 bits of the fifth byte of the EDP (EDP4).
NextCKId	14	uimsbf	EDP5[7:0] EDP6[7:2]	ID of the next content key to be used. The high 8 bits of NextCKId should be in the sixth byte of the EDP (EDP5), and the low 6 bits of NextCKId should be in the high 6 bits of the seventh byte of the EDP (EDP6).
NextCKType	2	uimsbf	EDP6[1:0]	Type of the next content key to be used, which is defined as follows: 00b: unicast. The next content key to be used is generated using derivation. 01b: multicast. The next content key to be used is distributed using a KDP. Others: reserved. NextCKType should be in the low 2 bits of the seventh byte of the EDP (EDP7).
ID_A	48	uimsbf	EDP7[7:0]... EDP12[7:0]	Unique ID of the AV stream transmitter A, which uses binary coding. The first byte of ID_A should be in the eighth byte of the EDP (EDP7), and the 2nd to 6th bytes of ID_A should be arranged in EDP8 to KDP12.
EncAlgorithm	4	uimsbf	EDP13[7:4]	Content encryption algorithm and mode, which are defined as follows: 0001b: SM4-CTR; Others: reserved. In SM4-CTR mode, the low 64 bits of the counter used for encryption, CtrLow, is 0. The counter is incremented by 1 each time a 16-byte AV stream data block is encrypted. The high 64 bits of the counter used for encryption is CtrLHigh. The initial CtrHigh can be generated based on a random number. When a new EDP is generated, CtrHigh is incremented by 1 based on the previous CtrHigh. EncAlgorithm should be in the high 4 bits of the 14th byte of the EDP (EDP13).
CtrHigh	64	uimsbf	EDP13[3:0] EDP14[7:0]... EDP20[7:0] EDP21[7:4]	High 64 bits of the counter used for encryption. CtrHigh should be unique. The high 4 bits of the first byte of CtrHigh should be in the low 4 bits of the 14th byte of the EDP (EDP13), the low 4 bits of the first byte of CtrHigh should be in the high 4 bits of the 15th byte of the EDP (EDP14), and the 2nd to 8th bytes of CtrHigh should be arranged from the low 4 bits of the 15th byte of the EDP (EDP14) to the high 4 bits of the 22nd byte of the EDP (EDP21).
Reserved	20	uimsbf	EDP21[3:0] EDP22[7:0] EDP23[7:0]	Reserved

When receiving an EDP, the receiver uses CurCKId, CurCKType, NextCKId, and NextCKType as the judgment conditions. If the four fields do not change, the receiver does not process the packet again (except CtrHigh). If other fields except CtrHigh change, the receiver updates the four fields to ensure that the EDP can be correctly processed.

In multicast scenarios, to achieve smooth key switchover, the transmitter must send the next content key to authorized receiver through a KDP before switching the content key.

When the content key requires an update, the transmitter updates NextCKId in the EDP.

Upon receiving the EDP, the receiver derives the content key based on NextCKType and NextCKId or obtains the content key through the KDP.

When a new content key needs to be applied, the transmitter switches CKId in the EDP and uses the content key corresponding to CKId to encrypt content. The receiver then updates the content key used for decryption based on CKId in the new EDP, thereby achieving smooth content key switchover.

8.5 Encrypted Content Transmission Through the GPMI

The GPMI standard defines the structure of a video frame, which consists of a vertical blanking region, a horizontal blanking region, and an active video region.

Figure 7 shows an example of an encrypted GPMI video frame.

EDPs and KDPs are transmitted in the vertical blanking region. Specifically, EDPs are transmitted immediately after the vertical blanking packet (VBP) of each frame, while KDPs are transmitted right after EDPs and before the next horizontal blanking packet (HBP). This transmission mechanism enables frame-level content protection for AV stream packets.

KDPs must be continuously transmitted for 600 to 1200 frames through the GPMI when encryption is initiated and the content key is updated. EDPs are transmitted in each frame during encryption, also through the GPMI. KDPs and EDPs use the cyclic redundancy check (CRC) bit for verification. To enhance the reliability of EDP transmission, the same EDP can be transmitted three times, while the KDP of each receiver must be transmitted at least once.

Each frame of active video data is transmitted via active video packets (AVPs). For encrypted AVPs, the content protection (CP) flag bit must be set to 1.

Audio sample packets (ASPs) are transmitted in the blanking region to carry audio data. Similarly, for encrypted ASPs, the content protection (CP) flag bit must be set to 1.

The SM4-CTR algorithm is used to encrypt audio and video packets. After CtrHigh is initialized or updated, CtrLow is initialized to 0. CtrHigh and CtrLow form a 16-byte counter. The content key and counter are used to encrypt audio and video data in AVP and ASP packets.

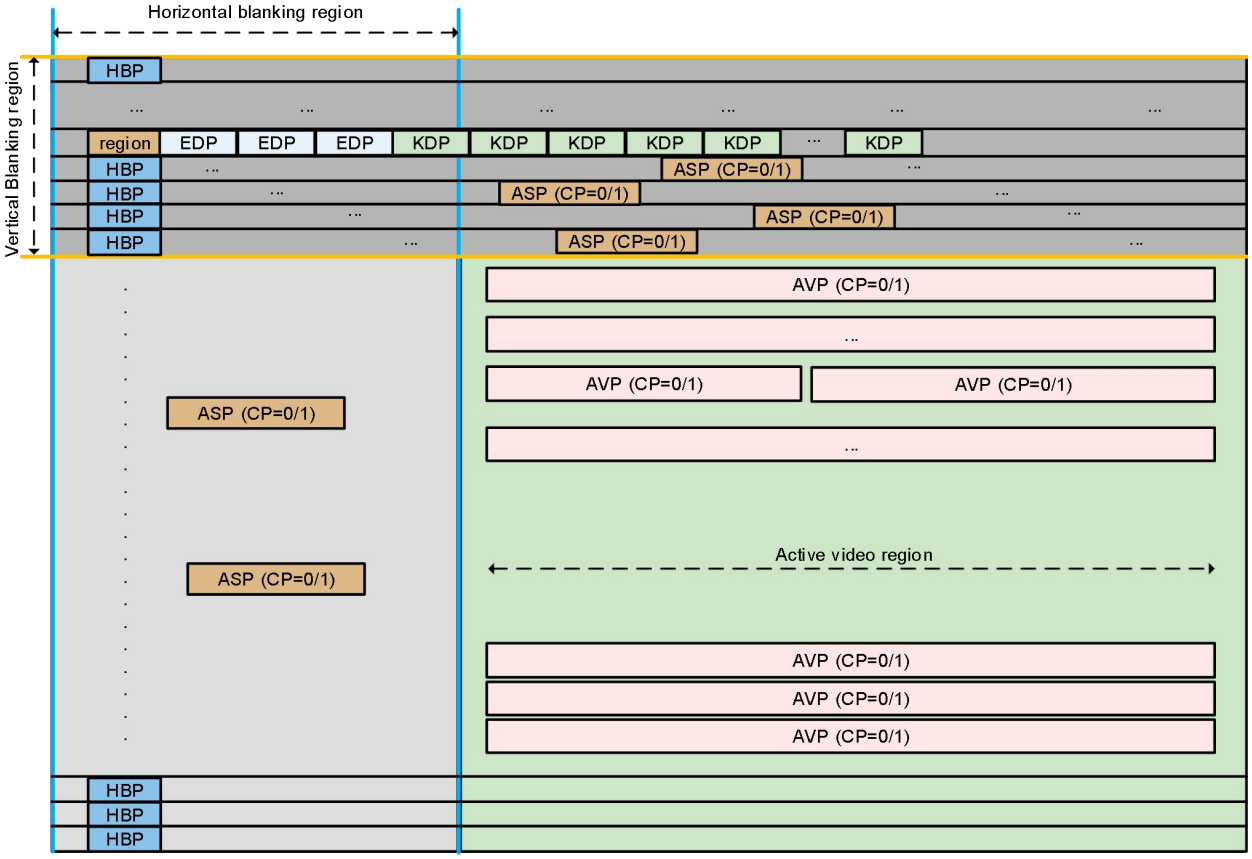

Figure 8 Example of an encrypted video frame

9 Trust System

The ADCP public key infrastructure (PKI) consists of the two-level certificate authority (CA).

Figure 8 illustrates the trust system.

Root CA: self-signed root certificate RootCert.

Device CA: issued by the root CA and used to sign a device certificate.

Device certificate: signed by the device CA, including device information such as the unique device ID and security level.

CRL CA: issued by the root CA and used to sign a CRL.

CRL: issued by the CRL CA and used for device revocation.

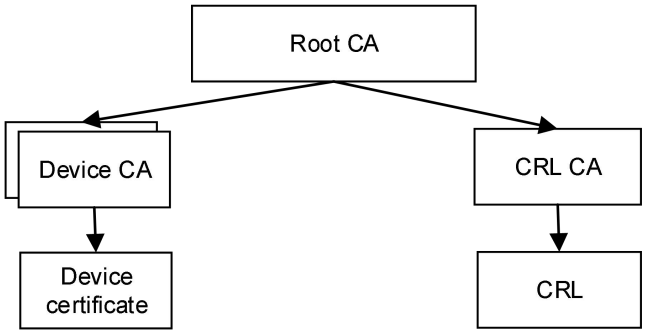

Figure 9 Trust system

The certificate format complies with the X.509v3 standard, uses DER encoding, and supports the signature algorithm combining SM3 with SM2. For details, see Appendix F.

The CRL complies with the X.509v2 CRL standard and uses DER encoding. For details, see Appendix F.5.

Before delivery, the AV stream transmitter comes pre-installed with a root CA certificate, a CRL CA certificate, and the latest CRL, which are used to authenticate the AV stream receiver. The AV stream receiver, on the other hand, is built in with a device CA certificate, a device private key, and the corresponding device certificate. Implements should ensure the secure distribution and provisioning of certificate information, with special attention to preventing the disclosure of the device private key to any third party. For details, see Appendix G.

Appendix A

(Normative)

Cryptographic Algorithm for the Authentication Algorithm Suite

For details, see Table A-1 Cryptographic algorithm for the authentication algorithm suite.

Table A-1 Cryptographic algorithm for the authentication algorithm suite

No.	High 4 Bits of AlgID	Elliptic Curve Algorithm	Signature Algorithm	Key Agreement Algorithm	Hash Algorithm	Message Authentication Algorithm	Key Derivation Algorithm	Content Key Encryption Algorithm
1	0001b	SM2, see GB/T 32918.5.	SM2, see GB/T 32918.2. The distinguishable identifier is the default value 1234567812345678.	ECDHE	SM3, see GB/T 32905.	HMAC-SM3	HKDF-SM3	SM4-CTR. For details about the SM4 algorithm, see GB/T 32907. For details about the CTR mode, see GB/T 17964.

Appendix B

(Normative)

Data Security Requirements

For details, see Table B-1 Sensitive data security requirements. Table B-1 describes the integrity and confidentiality requirements of data involved in the system and whether the data can be saved to local non-volatile storage.

Table B-1 Sensitive data security requirements

Data	Integrity Protection	Confidentiality Protection	Allow Saving to Local Non-volatile Storage
PrK_A, PrK_B	Yes	Yes	Yes
RootCert	Yes	No	Yes
DeviceCert, SubCACert	Yes	No	Yes
CRL, CRLSubCACert	Yes	No	Yes
CRL_ThisUpdate_A, CRL_ThisUpdate_B	Yes	No	No
AIR	Yes	Yes for Km, no for others	Yes
ID	Yes	No	Yes
Random_A, Random_B	Yes	No	No
DH_A, DH_B	Yes	Yes	No
DHPK_A, DHPK_B	Yes	No	No
DHSK	Yes	Yes	No
KHMAC_A, KHMAC_B, KHMAC_CRL	Yes	Yes	No
CK	Yes	Yes	No
CKId	Yes	No	No
CKEK	Yes	Yes	No
ECKCtr	Yes	No	No
CtrHigh, CtrLow	Yes	No	No

Appendix C

(Informative)

External System Interface

C.1 CRL Transfer

The CRL comes from the CRL CA and can be updated by external systems.

External systems shall transfer the CRL based on the algorithm suite supported by the device. Table C-1 describes the syntax format of the CRL transfer interface.

Table C-1 CRL transfer interface

Field Syntax	Number of Bits	Type	Description
ADCP_CRLUpdate(){			
Version=0x01	8	uimsbf	Interface protocol version, which is 0x01
MsgID=0x02	8	uimsbf	Message ID
Len[15..0]	16	uimsbf	Data length following this field, in bytes.
AlgNumber	8	uimsbf	Number of algorithm suites supported by the device. The current value is 1.
CRLList(){			
for (i=0;i<ADCP_AlgNumber;i++){			
CRL_Length[23..0]	24	uimsbf	CRL length
CRL[CRL_Length*8-1..0]	CRL_Length*8	uimsbf	Certificate revocation list
Cert_Length[15..0]	16	uimsbf	Length of the CRL CA certificate, in bytes
CRLSubCACert[Cert_Length*8-1..0]	Cert_Length*8	uimsbf	CRL CA certificate
}			
}			
}			

After obtaining the CRL, the ADCP device updates the CRL after verifying the validity of the CRL. If the CRL of the ADCP device is updated, the ADCP device checks whether all peer authentication devices are revoked. If a device is revoked, the ADCP device marks the authentication status of this device as failed. Figure C-1 shows the CRL update process.

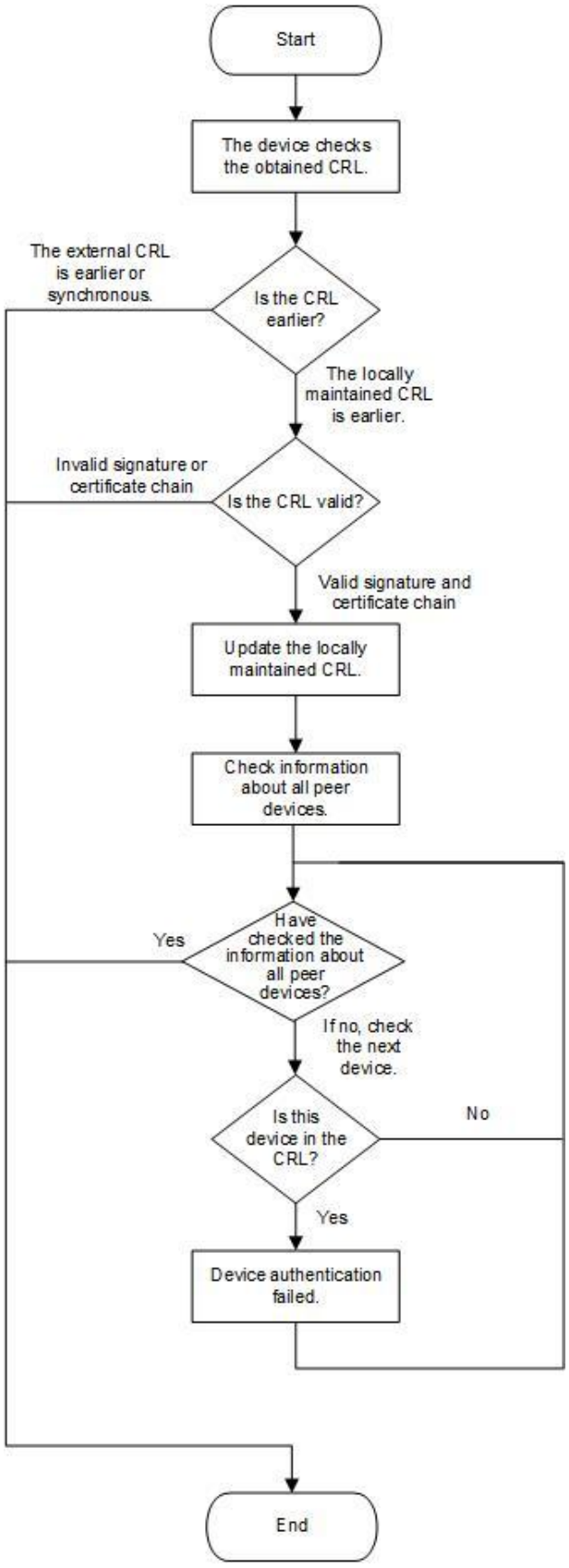

Figure C-1 CRL update process

C.2 Rights Control Policy Transfer Interface

The rights control policy transfer interface is used by external systems to send a rights control policy to the ADCP authorization module on an AV stream transmitter. Table C-2 describes the syntax format of this interface.

Table C-2 Parameters of the rights control policy transfer interface

Field Syntax	Number of Bits	Type	Description
ADCP_RightsControlPolicy(){			
Version=0x01	8	uimsbf	Interface protocol version, which is 0x01
MsgID=0x05	8	uimsbf	Message ID
Len[15..0]	16	uimsbf	Data length following this field, in bytes.
ADCP_VersionRequired[7..0]	8	uimsbf	Minimum protocol version number supported
ADCP_SecurityLevel[7..0]	8	uimsbf	Lowest security level supported
}			

C.3 Obtaining the ID List of AV Stream Receivers

The ADCP device provides an interface for obtaining the list of AV stream receivers. Based on this interface, external systems can obtain the receiver ID list from the AV stream transmitter. If the device has a device certificate, the device certificate chain and digital signature can be provided to ensure the authenticity of the obtained information.

Table C-3 describes the syntax format of the request for obtaining the ID list of AV stream receivers.

Table C-3 Syntax format of the request for obtaining the receiver ID list

Field Syntax	Number of Bits	Type	Description
ADCP_SinkIDReq(){			
Version=0x01	8	uimsbf	Interface protocol version, which is 0x01
MsgID=0x03	8	uimsbf	Message ID
Len[15..0]	16	uimsbf	Data length following this field, in bytes.
Nonce[127..0]	128	uimsbf	A randomly generated number
}			

Table C-4 describes the syntax format of the response from the ADCP device. If the response message contains a signature, the external system shall check the validity of the certificate chain and the correctness of the digital signature.

Table C-4 Syntax format of the response for obtaining the receiver ID list

Field Syntax	Number of Bits	Type	Description
ADCP_SinkIDRsp(){			
Version=0x01	8	uimsbf	Interface protocol version, which is 0x01
MsgID=0x04	8	uimsbf	Message ID

Len[15..0]	16	uimsbf	Data length following this field, in bytes.
ADCP_SinkIDInfo(){			
SinkIDNumber[7..0]	8	uimsbf	Number of AV stream receivers
SinkIDList(){			
for (i=0;i<SinkIDNumber;i++){			
AlgID[7..0]	8	uimsbf	Algorithm suite negotiated with the receiver device
DeviceSerialNumber[159..0]	160	uimsbf	Serial number of the receiver device certificate
SubCASerialNumber[159..0]	160	uimsbf	Sub-CA serial number of the receiver device certificate
ProductModelID[31..0]	32	uimsbf	Product ID of the receiver device
Version[7..0]	8	uimsbf	Protocol version number currently used by the receiver device
SecurityLevel[7..0]	8	uimsbf	Security level of the receiver device
}			
}			
}			
Nonce[127..0]	128	uimsbf	A randomly generated number
AttestationFlag			0x01: provides a signature. 0x00: does not provide a signature.
if(AttestationFlag==0x01){			
DeviceCert_length[15..0]	16	uimsbf	Device certificate length, in bytes
DeviceCert[DeviceCert_Length*8-1..0]	DeviceCert_ Length*8	uimsbf	Device certificate
SubCACert_Length[15..0]	16	uimsbf	Sub-CA certificate length of the device, in bytes
SubCACert[SubCACert_Length*8-1..0]	SubCACert_ Length*8	uimsbf	Sub-CA certificate of the device
AttestationValue_Len	8	uimsbf	AttestationValue length, in bytes
AttestationValue[AttestationValue_ Len*8-1..0]	AttestationValue _Len*8	uimsbf	Digital signature. The signature is signed by the private key in the device certificate. The signature algorithm is the same as the certificate signature algorithm. The signature includes all data exchanged in the protocol. Currently, the signature includes all fields from Version to SubCACert in ADCP_SinkIDReq and ADCP_SinkIDRsp.
}			
}			

C.4 ADCP Information Obtaining

The ADCP device provides an ADCP information obtaining interface for external systems to obtain device information, including the supported algorithm ID, ADCP version, and working status. If the device has a device certificate, the device certificate chain and digital signature can be provided to ensure the authenticity of the device information. Table C-5 describes the syntax format of an ADCP information obtaining request.

Table C-5 ADCP information obtaining request

Field Syntax	Number of Bits	Type	Description
ADCP_StatusReq(){			
Version=0x01	8	uimsbf	Interface protocol version, which is 0x01
MsgID=0x00	8	uimsbf	Message ID
Len[15..0]	16	uimsbf	Data length following this field, in bytes.
Nonce[127..0]	128	uimsbf	A randomly generated number
}			

Table C-6 describes the syntax format of an ADCP information obtaining response. If the response message contains a signature, the external system shall check the validity of the certificate chain and the correctness of the digital signature.

Table C-6 ADCP information obtaining response

Field Syntax	Number of Bits	Type	Description
ADCP_StatusRsp(){			
Version=0x01	8	uimsbf	Interface protocol version, which is 0x01
MsgID=0x01	8	uimsbf	Message ID
Len[15..0]	16	uimsbf	Data length following this field, in bytes.
ADCP_StatusInfo(){			
ADCP_Version=0x01	8	uimsbf	Supported highest ADCP version. The current version is 0x01.
AlgSupport	8	uimsbf	Supported algorithm suite. The current value is 0x01.
Status	8	uimsbf	0x01 indicates that the ADCP is activated, and 0x00 indicates that the ADCP is not activated.
}			
Nonce[127..0]	128	uimsbf	A randomly generated number
AttestationFlag	8	uimsbf	0x01: provides a signature. 0x00: does not provide a signature.
if(AttestationFlag==0x01){			
DeviceCert_length[15..0]	16	uimsbf	Device certificate length, in bytes
DeviceCert[DeviceCert_Length*8-1..0]	DeviceCert_Length*8	uimsbf	Device certificate
SubCACert_Length[15..0]	16	uimsbf	Sub-CA certificate length of the device, in bytes
SubCACert[SubCACert_Length*8-1..0]	SubCACert_Length*8	uimsbf	Sub-CA certificate of the device
AttestationValue_Len	8	uimsbf	AttestationValue length
AttestationValue[AttestationValue_ Len*8-1..0]	AttestationValue_ Len*8	uimsbf	Digital signature. The signature is signed by the private key in the device certificate. The signature algorithm is the same as the certificate signature algorithm. The signature includes all data exchanged in the protocol. Currently, the signature includes

			all fields from Version to SubCACert in ADCP_StatusReq and ADCP_StatusRsp.
}			
}			

Appendix D

(Informative)

Device Security Level Definition

For details about the device security level definition, see Table D-1 Security level.

Table D-1 Security level

Security Level	Random Number	Secure Storage	Cryptographic Algorithm	Stream Protection
L1	Pseudo-random number	Encryption and decryption implemented by software, supporting confidentiality and integrity protection	Implementation by software	Implementation by software
L2	Hardware true random number generator, meeting the GM/T 0005 requirements	Implemented in the trusted execution environment, supporting confidentiality and integrity protection	Implemented in the trusted execution environment or by hardware	Implemented in the trusted execution environment or by hardware
L3	Same as L2	Secure storage by hardware, supporting confidentiality and integrity protection	Implemented by hardware algorithms, supporting fault injection prevention, side channel attack prevention, and physical attack prevention	Implemented by hardware and cannot be read by software

Appendix E

(Informative)

Example of the Content Key Distribution Mechanism

E.1 Authentication Information

The AV stream transmitter generates an encryption description packet and a key distribution packet based on the rights control policy of the AV stream defined in chapter 7, and transmits the encryption description packet and the key distribution packet to the AV stream receiver together with the AV stream.

The AV stream transmitter acts as the authentication initiator A, and the AV stream receiver as the authentication responder B. The following is an example of authentication information:

1. Information about authentication initiator A:

ID_A:

0x11, 0x22, 0x33, 0x44, 0x55, 0x66

Km:

0x3E, 0xC8, 0x11, 0x05, 0x10, 0x27, 0x59, 0x39,

0xFA, 0xBB, 0x7F, 0x1B, 0xC5, 0x7A, 0x44, 0xFF,

0x69, 0xBF, 0x47, 0x64, 0x2F, 0x5C, 0x99, 0xBE,

0x58, 0xA7, 0x3A, 0x18, 0x0C, 0x6A, 0x32, 0x0D

Random_A:

0xE1, 0x62, 0x9A, 0xF6, 0xA5, 0xFC, 0x3D, 0xE9,

0xC8, 0x96, 0x85, 0x65, 0x02, 0x10, 0x2E, 0x39

2. Information about the authentication responder B:

ID_B:

0x11, 0x22, 0x33, 0x44, 0x55, 0x67

Km:

0x3E, 0xC8, 0x11, 0x05, 0x10, 0x27, 0x59, 0x39,

0xFA, 0xBB, 0x7F, 0x1B, 0xC5, 0x7A, 0x44, 0xFF,

0x69, 0xBF, 0x47, 0x64, 0x2F, 0x5C, 0x99, 0xBE,

0x58, 0xA7, 0x3A, 0x18, 0x0C, 0x6A, 0x32, 0x0D

Random_B:

0x3E, 0x32, 0x35, 0xA3, 0xEF, 0xED, 0x78, 0xD6,

0xEE, 0x62, 0xE0, 0x1C, 0xC2, 0x3F, 0xEE, 0xB8

E.2 Unicast Content Key Derivation

1. Unicast encryption description packet:

0000:02 01 15 00 00 00 00 11 22 33 44 55 66 10 10 20

0010:30 40 50 60 70 80 00 00

Field description:

Type: 0x02

Version: 0x01

Len: 0x15

CurCKId: 0x0

CurCKType: 0x0

NextCKId: 0x0

NextCKType: 0x0

ID_A: 0x11, 0x22, 0x33, 0x44, 0x55, 0x66

EncAlgorithm: 0x01

CtrHigh: 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08

Reserved: 0x0

2. Unicast content key derivation:

CK=KDF(Km, Random_A||Random_B||ID_A||ID_B||CKId, "Unicast Content Key", 128)=

0xA7, 0xAE, 0x0C, 0x90, 0x45, 0x58, 0x4F, 0x32,

0x34, 0x3F, 0xF8, 0xA2, 0x29, 0xE4, 0xF2, 0xD4

E.3 Unicast Content Key Switchover

1. Unicast encryption description packet before the switchover:

0000:02 01 15 00 00 00 04 11 22 33 44 55 66 10 10 20

0010:30 40 50 60 80 80 00 00

Field description:

Type: 0x02

Version: 0x01

Len: 0x15

CurCKId: 0x0

CurCKType: 0x0

NextCKId: 0x1

NextCKType: 0x0

ID_A: 0x11, 0x22, 0x33, 0x44, 0x55, 0x66

EncAlgorithm: 0x01

CtrHigh: 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x08, 0x08

Reserved: 0x0

2. Current unicast content key and the next unicast content key to be switched to:

Current CK=KDF (Km, Random_A||Random_B||ID_A||ID_B||CKId,"Unicast Content Key", 128)=

0xA7, 0xAE, 0x0C, 0x90, 0x45, 0x58, 0x4F, 0x32,

0x34, 0x3F, 0xF8, 0xA2, 0x29, 0xE4, 0xF2, 0xD4

Next CK=KDF (Km, Random_A||Random_B||ID_A||ID_B||CKId,"Unicast Content Key", 128)=

0x06, 0x5A, 0x1E, 0xE8, 0xFC, 0x31, 0xDA, 0x4E,

0x48,0x4E, 0x95, 0xB3, 0x83, 0x9D, 0xA6, 0xDA

3. Unicast encryption description packet during the switchover:

0000:02 01 15 00 04 00 04 11 22 33 44 55 66 10 10 20

0010:30 40 50 60 80 90 00 00

Field description:

Type: 0x02

Version: 0x01

Len: 0x15

CurCKId: 0x01

CurCKType: 0x0

NextCKId: 0x1

NextCKType: 0x0

ID_A: 0x11, 0x22, 0x33, 0x44, 0x55, 0x66

EncAlgorithm: 0x01

CtrHigh: 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x08, 0x09

Reserved: 0x0

4. Unicast content key during the switchover:

CK=KDF(Km, Random_A||Random_B||ID_A||ID_B||CKId, "Unicast Content Key", 128) =

0x06, 0x5A, 0x1E, 0xE8, 0xFC, 0x31, 0xDA, 0x4E,

0x48, 0x4E, 0x95, 0xB3, 0x83, 0x9D, 0xA6, 0xDA

E.4 Multicast Content Key Generation

1. Multicast encryption description packet:

0000:02 01 15 00 05 00 05 11 22 33 44 55 66 10 00 10

0010:20 30 40 50 60 70 00 00

Field description:

Type: 0x02

Version: 0x01

Len: 0x15

CurCKId: 0x01

CurCKType: 0x01

NextCKId: 0x01

NextCKType: 0x01

ID_A: 0x11, 0x22, 0x33, 0x44, 0x55, 0x66

EncAlgorithm: 0x01

CtrHigh: 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07

Reserved: 0x0

2. Key distribution packet:

0000:01 01 29 00 04 11 22 33 44 55 67 00 01 02 03 04

0010:05 06 07 08 09 0A 0B 0C 0D 0E 0F 22 11 0A 8C A6

0020:2F D1 12 D1 77 1E DD 40 7C 31 28 00

Field description:

Type: 0x01

Version: 0x01

Len: 0x29

CKId: 0x01

Reserved: 0x0

ID_B: 0x11, 0x22, 0x33, 0x44, 0x55, 0x67

ECKCtr:

0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,

0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F

ECK:

0x22, 0x11, 0x0A, 0x8C, 0xA6, 0x2F, 0xD1, 0x12,

0xD1, 0x77, 0x1E, 0xDD, 0x40, 0x7C, 0x31, 0x28

Reserved: 0x0

3. Multicast content key generation:

CKEK=KDF(Km, Random_A||Random_B||ID_A||ID_B, "Content Key Encryption Key", 128) =

0xE1, 0x56, 0x00, 0x51, 0x9A, 0xD9, 0xD4, 0x45,

0x70, 0x37, 0x72, 0x78, 0x1D, 0x9C, 0x65, 0x48

CK= 0xAF, 0x1F, 0x4D, 0x5C, 0xF7, 0x2E, 0x49, 0x44,

0xC1, 0xD6, 0x5B, 0x3A, 0x39, 0x5E, 0xA5, 0xBA

E.5 Multicast Content Key Switchover

1. Multicast encryption description packet before the switchover:

0000:02 01 15 00 05 00 09 11 22 33 44 55 66 10 00 10

0010:20 30 40 50 70 70 00 00

Field description:

Type: 0x02

Version: 0x01

Len: 0x15

CurCKId: 0x01

CurCKType: 0x01

NextCKId: 0x02

NextCKType: 0x01

ID_A: 0x11, 0x22, 0x33, 0x44, 0x55, 0x66

EncAlgorithm: 0x01

CtrHigh: 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x07, 0x07

Reserved: 0x0

2. Multicast key distribution packet of the content key to be switched:

0000:01 01 29 00 08 11 22 33 44 55 67 00 01 02 03 04

0010:05 06 07 08 09 0A 0B 0C 0D 0E 0F 52 91 36 A0 FA

0020:13 F6 EF D3 DC F7 7B F8 58 CD 2C 00

Field description:

Type: 0x01

Version: 0x01

Len: 0x29

CKId: 0x02

Reserved: 0x0

ID_B: 0x11, 0x22, 0x33, 0x44, 0x55, 0x67

ECKCtr:

0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,

0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F

ECK:

0x52, 0x91, 0x36, 0xA0, 0xFA, 0x13, 0xF6, 0xEF,

0xD3, 0xDC, 0xF7, 0x7B, 0xF8, 0x58, 0xCD, 0x2C

Reserved:

3. Multicast content key to be switched:

CKEK=KDF(Km, Random_A||Random_B||ID_A||ID_B, "Content Key Encryption Key", 128) =

0xE1, 0x56, 0x00, 0x51, 0x9A, 0xD9, 0xD4, 0x45,

0x70, 0x37, 0x72, 0x78, 0x1D, 0x9C, 0x65, 0x48

CK= 0xDF, 0x9F, 0x71, 0x70, 0xAB, 0x12, 0x6E, 0xB9,

0xC3, 0x7D, 0xB2, 0x9C, 0x81, 0x7A, 0x59, 0xBE

4. Multicast encryption description packet during the switchover:

0000:02 01 15 00 09 00 09 11 22 33 44 55 66 10 00 10

0010:20 30 40 50 70 80 00 00

Field description:

Type: 0x02

Version: 0x01

Len: 0x15

CurCKId: 0x02

CurCKType: 0x01

NextCKId: 0x02

NextCKType: 0x01

ID_A: 0x11, 0x22, 0x33, 0x44, 0x55, 0x66

EncAlgorithm: 0x01

CtrHigh: 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x07, 0x08

Reserved: 0x0

5. Multicast key distribution packet during the switchover:

0000:01 01 29 00 08 11 22 33 44 55 67 00 01 02 03 04

0010:05 06 07 08 09 0A 0B 0C 0D 0E 0F 52 91 36 A0 FA

0020:13 F6 EF D3 DC F7 7B F8 58 CD 2C 00

Field description:

Type: 0x01

Version: 0x01

Len: 0x29

CKId: 0x02

Reserved: 0x0

ID_B: 0x11, 0x22, 0x33, 0x44, 0x55, 0x67

ECKCtr:

0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,

0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F

ECK:

0x52, 0x91, 0x36, 0xA0, 0xFA, 0x13, 0xF6, 0xEF,

0xD3, 0xDC, 0xF7, 0x7B, 0xF8, 0x58, 0xCD, 0x2C

Reserved:

6. Multicast content key during the switchover:

CKEK=KDF(Km, Random_A||Random_B||ID_A||ID_B, "Content Key Encryption Key", 128) =

0xE1, 0x56, 0x00, 0x51, 0x9A, 0xD9, 0xD4, 0x45,

0x70, 0x37, 0x72, 0x78, 0x1D, 0x9C, 0x65, 0x48

CK= 0xDF, 0x9F, 0x71, 0x70, 0xAB, 0x12, 0x6E, 0xB9,

0xC3, 0x7D, 0xB2, 0x9C, 0x81, 0x7A, 0x59, 0xBE

Appendix F

(Normative)

Certificate and CRL Format

F.1 Root CA Certificate Format

The format must comply with Table F-1 Root CA certificate format.

Table F-1 Root CA certificate format

Field	Description
Version	Version number. The current version is v3, and the integer value is 2.
SerialNumber	Certificate serial number
Signature	Signature algorithm: SM2-with-SM3 (OID: 1.2.156.10197.1.501), which shall comply with the provisions in GB/T 32918.2
Issuer	Issuer, which is the same as the Subject field
Validity	Validity period, which is 50 years from the date of signature
Subject	Subject, including the following attributes: — — Country/Region name (countryName) (for example, "CN") — — Organization name (oranizationName) (for example, "ADCP") — — Common name (commonName) (for example, "Root CA")
SubjectPublicKeyInfo	Subject public key information The SM2 algorithm is used, including: — — Algorithm identifier id-ecPublicKey OID 1.2.840.10045.2.1 — — SM2 curve identifier OID 1.2.156.10197.1.301 — — SM2 public key
BasicConstraints extension	Basic constraints extension, which is required and critical. The CA field is set to TRUE.
KeyUsage extension	Key usage extension, which is required and critical. Only the keyCertSign bit is set.
Subject Key Identifier	Subject key identifier extension, which is required but not critical.
SignatureAlgorithmId	Signature algorithm identifier: SM3 with SM2, OID 1.2.156.10197.1.501
SignatureValue	Signature value

F.2 Device CA Certificate Format

The format must comply with Table F-2 Device CA certificate format.

Table F-2 Device CA certificate format

Field	Description
Version	Version number. The current version is v3, and the integer value is 2.
SerialNumber	Certificate serial number, which is a unique non-negative integer in the device certificate issued by the certificate center.
Signature	Signature algorithm: SM2-with-SM3 (OID: 1.2.156.10197.1.501), which shall comply with the provisions in GB/T 32918.2
Issuer	Issuer, which is the same as the Subject field of the root CA
Validity	Validity period, which is 20 years from the date when the certificate is generated. The value uses the UTCTime encoding format.
Subject	Subject, including the following attributes: — — Country/Region name (countryName) (for example, "CN") — — Organization name (organizationName) (for example, "ADCP") — — Common name (commonName) (for example, "Device CA 1")
SubjectPublicKeyInfo	Subject public key information The SM2 algorithm is used, including: — — Algorithm identifier id-ecPublicKey OID 1.2.840.10045.2.1 — — SM2 curve identifier OID 1.2.156.10197.1.301 — — SM2 public key
BasicConstraints extension	Basic constraints extension, which is required and critical. The CA field is set to TRUE. The PathLenConstraint field is set to 0.
KeyUsage extension	Key usage extension, which is required and critical. Only the keyCertSign bit is set.
Authority Key Identifier	Issuer key identifier extension, which is required but not critical.
Subject Key Identifier	Subject key identifier extension, which is required but not critical.
SignatureAlgorithmId	Signature algorithm identifier: SM3 with SM2, OID 1.2.156.10197.1.501
SignatureValue	Signature value

F.3 CRL CA Certificate Format

The format must comply with Table F-3 CRL CA certificate format.

Table F-3 CRL CA certificate format

Field	Description
Version	Version number. The current version is v3, and the integer value is 2.
SerialNumber	Certificate serial number, which is a unique non-negative integer in the device certificate issued by the certificate center.
Signature	Signature algorithm: SM2-with-SM3 (OID: 1.2.156.10197.1.501), which shall comply with the provisions in GB/T 32918.2
Issuer	Issuer, which is the same as the Subject field of the root CA
Validity	Validity period, which is 20 years from the date when the certificate is generated. The

	value uses the UTCTime encoding format.
Subject	Subject, including the following attributes: — — Country/Region name (countryName) (for example, "CN") — — Organization name (organizationName) (for example, "ADCP") — — Common name (commonName) (for example, "CRL CA 1")
SubjectPublicKeyInfo	Subject public key information The SM2 algorithm is used, including: — — Algorithm identifier id-ecPublicKey OID 1.2.840.10045.2.1 — — SM2 curve identifier OID 1.2.156.10197.1.301 — — SM2 public key
BasicConstraints extension	Basic constraints extension, which is required and critical. The CA field is set to TRUE. The PathLenConstraint field is set to 0.
KeyUsage extension	Key usage extension, which is required and critical. Only the cRLSign bit is set.
Authority Key Identifier	Issuer key identifier extension, which is required but not critical.
Subject Key Identifier	Subject key identifier extension, which is required but not critical.
SignatureAlgorithmId	Signature algorithm identifier: SM3 with SM2, OID 1.2.156.10197.1.501
SignatureValue	Signature value

F.4 Device Certificate Format

The format must comply with Table F-4 Device certificate format.

Table F-4 Device certificate format

Field	Description
Version	Version number. The current version is v3, and the integer value is 2.
SerialNumber	Certificate serial number, which is a unique non-negative integer in the device certificate issued by the certificate center.
Signature	Signature algorithm: SM2-with-SM3 (OID: 1.2.156.10197.1.501), which shall comply with the provisions in GB/T 32918.2
Issuer	Issuer, which is the same as the Subject field of the device CA that issues the certificate.
Validity	Validity period, which is 15 years from the date when the certificate is generated. The value uses the UTCTime encoding format.
Subject	Subject, including the following attributes: —— Country/Region name (countryName) —— Organization name (oranizationName) —— Common name (commonName), which is a string consisting of four parts separated by hyphens (-): Protocol version: two hexadecimal characters. The current version is 01. Product ID: eight hexadecimal characters, consisting of four characters of the Vendor ID and four characters of the Product ID, for example, 00010abd. Specifically, 0001 indicates the vendor ID, and 0abd indicates the specific product ID. Device type: one character. The options are 1, 2, and 3, indicating the AV stream

	transmitter, AV stream receiver, and AV stream transmitter and receiver, respectively. Security level: one character. The options are L1, L2, and L3, corresponding to 1, 2, and 3, respectively. Unique device ID: 12 hexadecimal characters, for example, 112233aabbcc. Example: 01-00010abd-2-1-112233aabbcc
SubjectPublicKeyInfo	Subject public key information The SM2 algorithm is used, including: — — Algorithm identifier id-ecPublicKey OID 1.2.840.10045.2.1 — — SM2 curve identifier OID 1.2.156.10197.1.301 — — SM2 public key
BasicConstraints extension	Basic constraints extension, which is required and critical. The CA field is set to FALSE.
KeyUsage extension	Key usage extension, which is required and critical. Only the digitalSignature bit is set.
Authority Key Identifier	Issuer key identifier extension, which is required but not critical.
SignatureAlgorithmId	Signature algorithm identifier: SM3 with SM2, OID 1.2.156.10197.1.501
SignatureValue	Signature value

F.5 CRL Format

The format must comply with Table F-5 CRL format.

Table F-5 CRL format

Field	Description
Version	Version number. The current version is v2, and the integer value is 1.
Signature	Signature algorithm: SM2-with-SM3 (OID: 1.2.156.10197.1.501), which shall comply with the provisions in GB/T 32918.2
IssuerName	Issuer, which is the same as the Subject field of the CRL CA
ThisUpdate	Issuance time, which uses the UTCTime encoding format, that is, before 2050.
NextUpdate	Next issuance time
Authority Key Identifier	Issuer key identifier extension, which is required but not critical.
UserCertificate	Revoked certificate, which is an integer. The default value is the certificate serial number. The value can also be the product ID.
RevocationDate	Revocation date, which uses the UTCTime encoding format.

UserCertificateType	Private CRL entry extension defined by ADCP, which is optional and critical. If the CRL does not contain the UserCertificateType extension, the certificate with the specified serial number in userCertificate is revoked. If the CRL contains the UserCertificateType extension, the ASN.1 syntax of UserCertificateType is as follows:
	id-pe-userCertificateType OBJECT IDENTIFIER::= {id-pe 34}
	id-pe::= {
	Iso(1) identified-organization(3) dod(6) internet(1) security(5)
	mechanisms(5) pkix(7) 1 }
	UserCertificateType::= ENUMERATED {
	revokedSerialNumber(0),

	revokedProductModel(1) } revokedSerialNumber: indicates that the certificate with the specified serial number in userCertificate is revoked. revokedProductModel: indicates that all certificates of the specified product model in userCertificate are revoked.
SignatureAlgorithm	Signature algorithm identifier: SM3 with SM2, OID 1.2.156.10197.1.501
SignatureValue	Signature value

Appendix G

(Informative)

Certificate Information Provisioning

G.1 Provisioning Reference Model

The ADCP trust center provides the ADCP device vendor with the production and distribution services of device key materials such as the ADCP device private key, device certificate, CRL, and CA certificate. The ADCP device vendor manages device key materials and imports them into the certificate provisioning service. Then, the certificate provisioning service places certificate information such as the device private key, device certificate, and CRL into the ADCP device. The certificate information is provisioned through the production line of the device. Figure G-1 shows the reference model for certificate information provisioning.

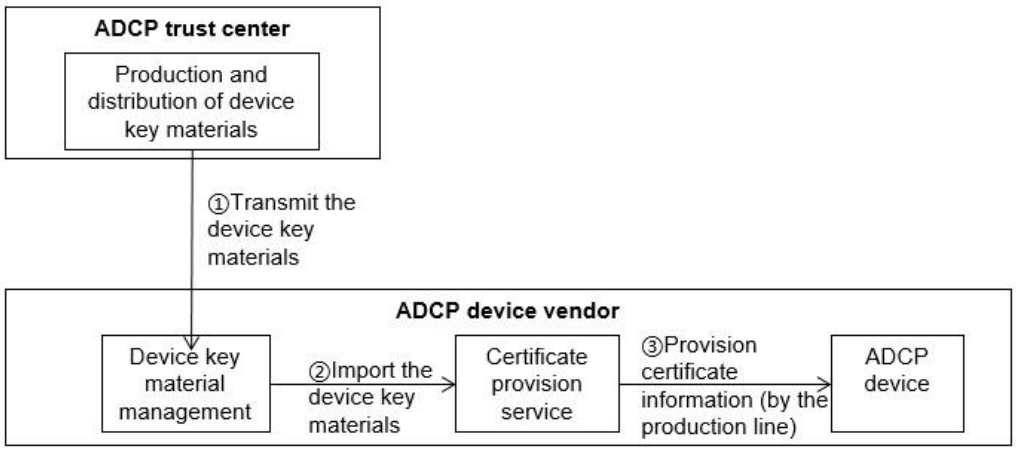

Figure G-1 Certificate information provisioning model

In Figure G-1, processes ①, ②, and ③ ensure the confidentiality of the device private key and the integrity of the certificate information. Process ③ ensures the consistency between the provisioned device private key and device certificate, and also ensures that a device certificate is used only for a unique device.

G.2 Protection of the Device Private Key

To protect the confidentiality of the device private key, the ADCP device vendor can take the following measures:

a) Set up a secure electronic storage place for storing the device private key.

b) Set access control for the stored device private key.

c) Store the device private key in a locked secure location or store it electronically in a secure manner.