Android PlayReady

Development Guide

Issue	**00B03**
Date	**2015-09-10**

Trademarks and Permissions

 , HISILICON, and other HiSilicon icons are trademarks of HiSilicon Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between HiSilicon and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

HiSilicon Technologies Co., Ltd.

Address: Huawei Industrial Base
Bantian, Longgang
Shenzhen 518129
People's Republic of China

Website: http://www.hisilicon.com

Email: support@hisilicon.com

About This Document

Overview

This document describes the working principle of PlayReady. This document also describes how to develop PlayReady on a HiSilicon Android platform and the precautions.

Product Versions

The following table lists the product versions related to this document.

Product Name	Product Version
Hi3716C	V2XX
Hi3719C	V1XX
Hi3719M	V1XX
Hi3716M	V4XX
Hi3798M	V1XX
Hi3798C	V2XX
HiSTBAndroid	V600R001
HiSTBAndroid	V600R002

Intended Audience

This document is intended for:

- Technical support engineers
- Software development engineers

Change History

Changes between document issues are cumulative. Therefore, the latest document issue contains all changes made in previous issues.

Issue 00B03 (2015-09-10)

This issue is the third draft release, which incorporates the following changes:

Chapter 3 PlayReady SW Development

Compilation instruction is added.

Test description is modified.

Section 3.4.5 is modified.

Issue 00B02 (2014-11-06)

This issue is the second draft release, which incorporates the following changes:

Descriptions of the SW version configuration file are modified.

The certificate encryption algorithm for factory production is changed to the AES-CTR mode.

Issue 00B01 (2014-06-16)

This issue is the first draft release.

Contents

Figures

Tables

1 Overview

1.1 Introduction to PlayReady

PlayReady is a new digital rights management (DRM) system of Microsoft and is the updated system of Windows media DRM (WMDRM). PlayReady is fully compatible with the WMDRM and provides content protection for the digital media.

Information about PlayReady products, technologies, documents, licenses, and support can be obtained from the official website of Microsoft PlayReady, that is, http://www.microsoft.com/playready/. Users can test the basic functions of PlayReady products using the test cases and streams provided by the official test website of Microsoft PlayReady, that is, http://playready.directtaps.net/.

The HiSilicon Android PlayReady solution supports playing of PIFF/Smooth Streaming media (file name: Manifest) using a web page and playing of ASF (extension name: pyv/pya) media locally.

The HiSilicon Android PlayReady solution has two versions: hardware (HW) and software (SW). The SW version provides basic and necessary protection for contents and certificates and has no special requirements on the chipset and scheme. Therefore, this version is applicable to scenarios where content providers do not have special requirements. The HW version employs the TrustZone hardware protection mechanism of HiSilicon chipsets and provides a high-level protection for certificates, encrypted/decrypted keys, and decrypted streams. Therefore, this version is applicable to scenarios where content providers have requirements on the TrustZone feature or Trust Video Path scenario.

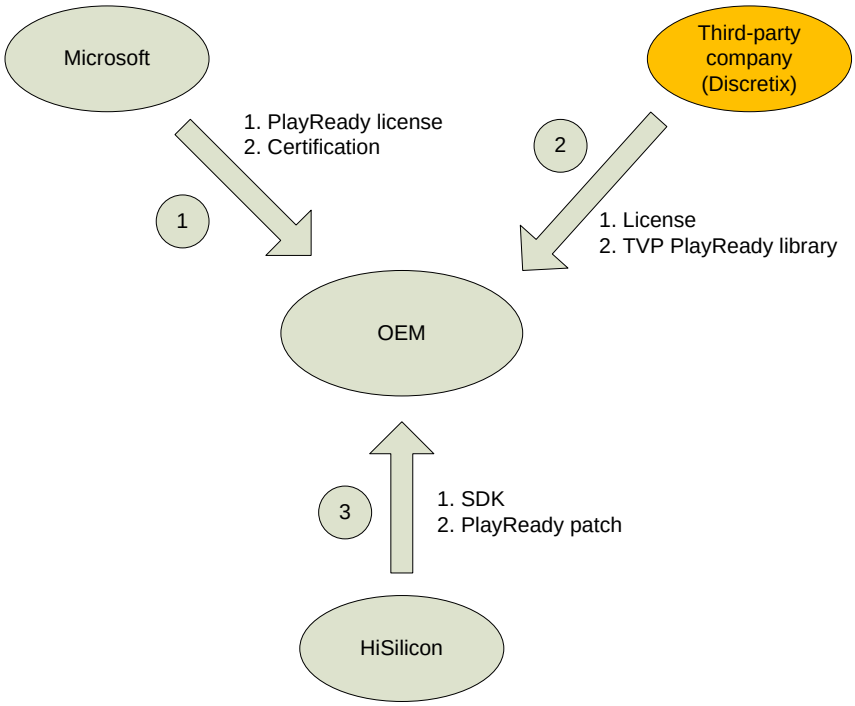
The HW version is a partner product provided by HiSilicon and a third-party company. The third-party company provides the PlayReady library and plug-ins meeting the Trust Video Path and Android DRM framework requirements based on the TrustZone platform provided by HiSilicon. To be an original equipment manufacturer (OEM), the manufacturer must obtain authorization of the PlayReady Final Product from Microsoft, authorization from the third-party company, and corresponding PlayReady library for the HW version. Currently, HiSilicon chooses Discretix (www.discretix.com) as the third-party company.

1.2 PlayReady Project Development Procedure

1.2.1 Cooperation Relationship for a Project

Figure 1-1 shows the cooperation relationship for a project.

Figure 1-1 Cooperation relationship for a project



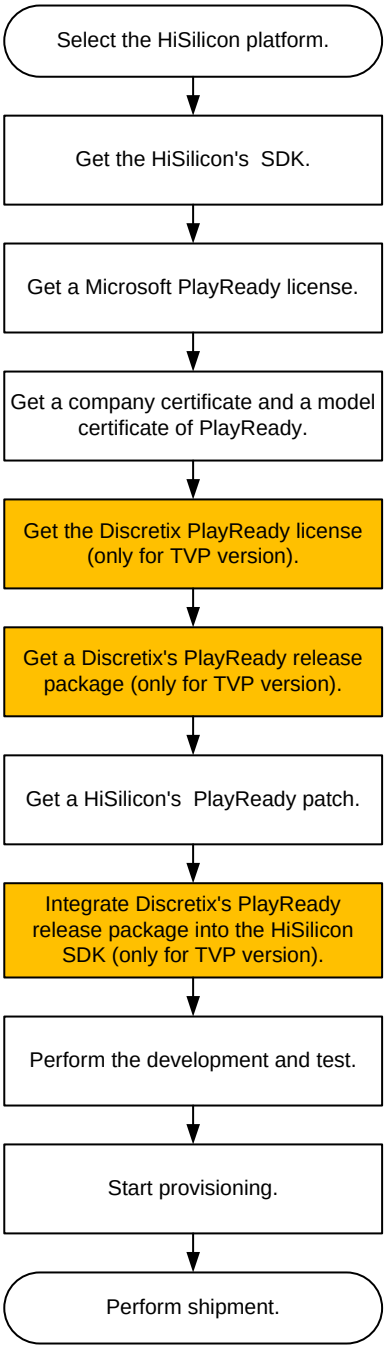
CAUTION

For the SW version, HiSilicon provides a patch package containing required library files. Users only need to obtain the licenses and certificates from Microsoft. For the HW version, besides the preceding files, users need to obtain the PlayReady library files and licenses from the third-party company (Discretix).

1.2.2 Project Development Procedure

Figure 1-2 shows the procedure to develop a project.

Figure 1-2 Project development procedure



To develop a project, do as follows:

Step 1 Select a HiSilicon platform. For the HW version, select a platform that supports the TrustZone feature.

Step 2 Obtain the Android software development kit (SDK) that supports PlayReady from HiSilicon.

Step 3 Obtain a PlayReady license and a PlayReady company certificate from Microsoft, and generate a model certificate.

Step 4 (Optional) For the HW version, obtain a PlayReady license from Discretix and a PlayReady released package that matches the HiSilicon platform and SDK.

Step 5 Obtain a PlayReady patch package matching the SDK from HiSilicon.

Step 6 (Optional) For the HW version, integrate the PlayReady files released by Discretix to the SDK provided by HiSilicon.

Step 7 An OEM develops a PlayReady application based on the reference PlayReady Demo APK.

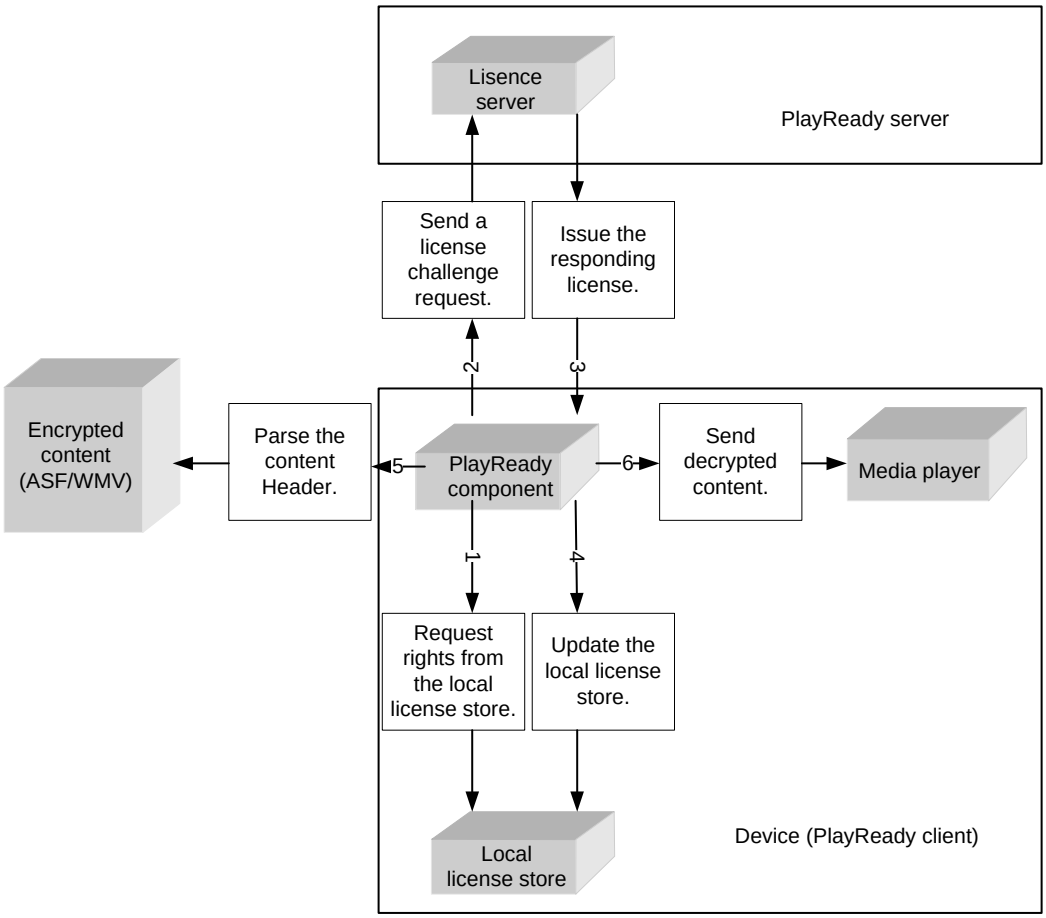
Step 8 Conduct mass production in a factory to finish burning of certificates and keys.

----End

1.3 Work Process of the PlayReady HW/SW Version

Figure 1-3 shows the PlayReady work process.

Figure 1-3 Work process of PlayReady



The work process is as follows:

Step 1 When a user attempts to play a file, the player queries PlayReady whether the file can be played.

Step 2 PlayReady searches the license library for a corresponding valid license.

Step 3 PlayReady sends a license request to the license server if no valid license is found in the license library.

Step 4 The license server returns a license containing a key for decrypting contents to PlayReady.

Step 5 PlayReady verifies and parses the license. PlayReady decrypts the key and puts the license to the license library.

Step 6 PlayReady decrypts the data packets and sends them to the player for decoding and playing.

----End

1.4 Device Certificate

The PlayReady device certificate is used for obtaining and decrypting stream licenses. OEMs can obtain this certificate from Microsoft.

1.4.1 Generating a Certificate

CAUTION

Microsoft has announced that only the final product company can apply for a new device certificate (the number of times that the final product company applies for a device certificate is not limited). As an intermediate product distribution company, HiSilicon can apply for a device certificate only once. The device certificate is used only for the internal test. If HiSilicon gives the device certificate to customers, it violates the PlayReady protocol. Therefore, the customers must apply for PlayReady certificates by themselves.

OEMs need to apply for a company certificate from Microsoft and generate a model certificate using the company certificate and tools as well as templates provided by Microsoft. The model certificate needs to be burnt to the device for obtaining and decrypting the stream license.

As PlayReady is compatible with the WMDRM, the PlayReady certificate includes the WMDRM certificate.

For details about how to apply for and generate certificates, see section 1.5 "Applying for and Using a PlayReady Certificate."

1.4.2 Security Level

Microsoft has defined the security levels of various certificates. For a non-commercial PlayReady certificate, the security level is 150; but for a commercial PlayReady certificate, the security level can reach 2000.

1.5 Applying for and Using a PlayReady Certificate

The process of applying for and using a PlayReady certificate is as follows:

Step 1 Generate **RequestFile** and **OEMPrivateKeyFile.xml** by using the sample tool **GenerateCompanyCertRequest.exe** in the Microsoft PlayReady SDK. Send **RequestFile** to Microsoft. **OEMPrivateKeyFile.xml** is saved by applicants in secret. Run the following command:

```
generatecompanycertrequest.exe -r:Request.xml -p:XXXOEMPrivateKeyFile.xml
-m:"Huawei Technologies Co."
```

Use P**layReady Device Development and Intermediate Product Distribution License CERTIFICATE REQUEST FORM** in EXHIBIT B of the PlayReady protocol to apply for a table and fill in the corresponding information in **Company Name** and **Effective Date and Agreement Number**.

Step 2 About ten working days later, Microsoft completes the processing of the certificate request and sends **DeviceCompanyCert.exe** to the applicants.

Step 3 Decompress **DeviceCompanyCert.exe** to obtain **NDResponse.crt**, **PDDACResponse.dat**, and **PRDACResponse.dat**. Only **PDDACResponse.dat** and **PRDACResponse.dat** are required.

Generate **UnsignedTemplate.xml**. **UnsignedTemplate.xml** must be saved in UTF-16 little endian format. You can save **UnsignedTemplate.xml** by using Windows Notepad. For details about the unsigned template file, see **playReady.chm** in PlayReady SDK. Refer to the following template sample in the **playReady.chm** file:

PlayReady Device Porting Kit v2.0 > **Getting Started** > **Obtaining Certificates** > **Creating a Model Certificate** > **Creating the Unsigned Template** > **PlayReady and WMDRM-PD Device Template Sample**

Or use the following settings:

```
<UNSIGNEDTEMPLATE>
    <NAME>Hi3716</NAME>
    <MANUFACTURER>HiSilicon</MANUFACTURER>
    <MAKE>UnknownMake</MAKE>
    <DISTRIBUTOR>WideWorldImporters</DISTRIBUTOR>
    <MODEL>Hi3716X</MODEL>
    <SECURITYLEVEL>2000</SECURITYLEVEL>
    <HARDWARE_VER_MAJOR>2</HARDWARE_VER_MAJOR>
    <HARDWARE_VER_MINOR>1</HARDWARE_VER_MINOR>
    <FIRMWARE_VER_MAJOR>1</FIRMWARE_VER_MAJOR>
    <FIRMWARE_VER_MINOR>3</FIRMWARE_VER_MINOR>

    <FEATURES>
        <CLOCK>2</CLOCK>
        <SECURECLOCK>
            <URL>http://go.microsoft.com/fwlink/?LinkID=149408</URL>

<PUBLICKEY>!CNhvvz1WaNV1AFUmetxkvm9iD4UrE9cnGUi!qcqdxMiXmD1*ikYGA==</PUBL
ICKEY>
      </SECURECLOCK>
      <METERING>1</METERING>
```

```
        <LICENSE_ACQ>1</LICENSE_ACQ>
        <LICENSE_SYNC>1</LICENSE_SYNC>
        <ENCRYPTION>1</ENCRYPTION>
        <SYMMETRIC_OPT>1</SYMMETRIC_OPT>
        <SUPPORT_REVOCATION>0</SUPPORT_REVOCATION>
    </FEATURES>

    <LIMITS>
        <MAXCHAINDEPTH>2</MAXCHAINDEPTH>
        <MAXLICENSESIZE>10240</MAXLICENSESIZE>
    <MAXHEADERSIZE>5120</MAXHEADERSIZE>
</LIMITS>

</UNSIGNEDTEMPLATE>
```

CAUTION

Remove the following fields in the template:

<TRANSMITTER>1</TRANSMITTER>

<RECEIVER>1</RECEIVER>

Step 4 To use the sample tool **generatemodelcert.exe**, run the following commands:

```
generatemodelcert.exe -d:PDDACResponse.dat  -b:PRDACResponse.dat
-u:UnsignedTemplate.xml -f:OEMPrivateKeyFile.xml -t:devcerttemplate.dat
-g:bgroupcert.dat -h:zgpriv.dat  -k:priv.dat
```

NOTE

Ensure that the previous files are in the current directory.

After **generatemodelcert.exe** is running, a message indicating success is displayed.

The following files are generated:

- **devcerttemplate.dat** and **priv.dat** (the certificate and private keys of the WMDRM)
- **bgroupcert.dat** and **zgpriv.dat** (the certificate and private keys of PlayReady)

The files **bgroupcert.dat**, **zgpriv.dat**, **devcerttemplate.dat**, and **priv.dat** are PlayReady certificates.

----End

CAUTION

The default security level of a certificate is 150, which is the lowest security level. After the license server (LS) adds some security requirements on the media files, the player cannot play the corresponding media files. Therefore, you need to apply for a license of a proper security level.

1.6 Private Keys and Security

The PlayReady security depends on the generated public and private key pair. The public key is stored in the device certificate and the private key needs to be properly stored by users.

CAUTION

The device certificates must be properly stored by OEMs.

1.7 Compliance Rules and Robustness Rules for Microsoft PlayReady

As required by Microsoft, the PlayReady final product licenses must comply with Compliance Rules and Robustness Rules.

CAUTION

The Compliance Rules and Robustness Rules file defines the output restrictions for audio and video media with different OPLs and must be abided by as the final product license. For example, an HDMI port cannot output decrypted audio or video media with the OPL exceeding 300.

You can obtain Compliance Rules and Robustness Rules by visiting http://www.microsoft.com/PlayReady/Licensing/compliance.mspx.

1.8 Secure Clock

A clock is required for licenses that limit the use of contents by time restrictions, such as beginning dates and expiration dates. A clock is particularly important in rental and subscription scenarios.

This component sets the system time to secure clock by connecting to the Microsoft secure clock service. When the system time is in reliable state, the device can gain access to the

media file by using PlayReady. When the system time is in unreliable state, the device cannot play the contents and obtain the license by using PlayReady.

Typically, the device updates the system clock to secure clock periodically.

1.9 Domains

PlayReady allows a set of devices to play all the media files bound to a domain as the members of the domain. Therefore, the media files in the domain are shared. A device can be a member of multiple domains.

By joining a PlayReady domain, a device can obtain the content bound to this domain. Therefore, the device can play all the media files bound to this domain.

The service that manages the membership in a PlayReady domain may restrict the number of devices in a domain. In such a case, the user must remove a device from the domain before adding another. A user that buys a replacement device should remove the old device from the domain.

1.10 Metering

In the ordering service of media contents, the metering function of PlayReady is used to report the times that users play the media to the owners of the contents. The fee is charged based on the play times. The metering function provides preference in some cases. For example, the royalty for online content providers is cut greatly. Usually, the fee of purchasing the play right of the content is lower than that of transferring the content permanently.

The process of reporting metering data is as follows:

Step 1 The device checks whether a local metering certificate with the metering ID (MID) of the media is available.

- If such metering certificate is available, go to Step 3.
- If no such metering certificate is available, the device applies from the metering certificate service for a required metering certificate.

Step 2 The device collects the metering data with the MID of the content, encrypts it, and sends it to the metering service URL provided by the metering certificate.

Step 3 The metering service confirms the metering report, notifies the device of this report, and clears the metering counter on the device.

If the metering data is in large amount, the device divides it into parts and sends them to the metering service URL by repeating the operations in Step 2 and Step 3.

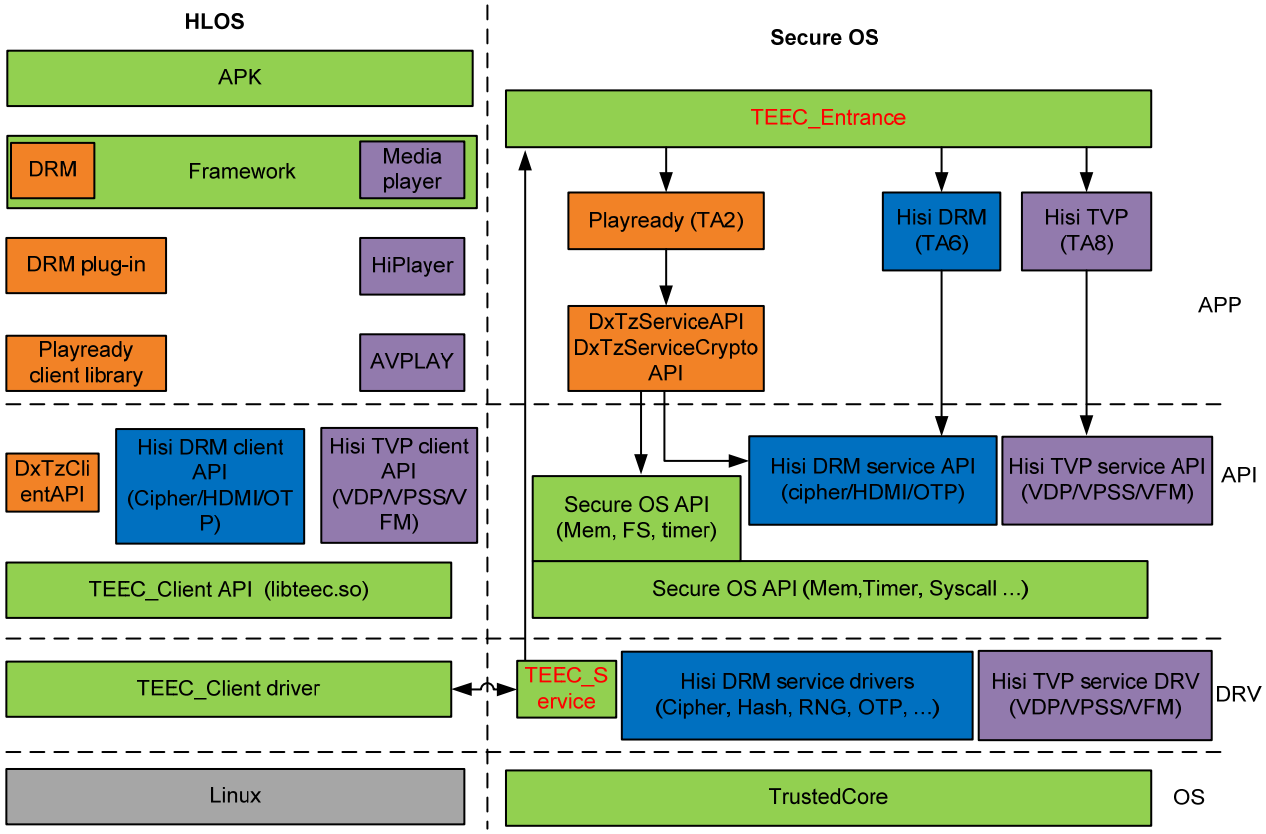
----End

2 PlayReady HW Development

2.1 Block Diagram of the Scheme

Figure 2-1 shows the block diagram of the PlayReady HW scheme.

Figure 2-1 Block diagram of the PlayReady HW scheme



The PlayReady HW scheme employs the TrustZone hardware protection mechanism of HiSilicon chipsets and provides a high-level protection for certificates, licenses, and decrypted data. Data decrypting and decoding are both performed in the secure domain.

As shown in Figure 2-1, the parts in orange indicate that data is decrypted to the secure memory; the parts in purple indicate that decrypted data is decoded, played, and displayed. The decrypting and decoding are interconnected through the framework layer of the Android platform.

2.2 Preparations

Before starting development, perform the following preparations:

Step 1 Obtain the PlayReady HW release package matching with the SDK from Discretix.

Step 2 Obtain the PlayReady HW patch package matching with the SDK from HiSilicon and put is in the Android SDK root directory.

Step 3 Decompress the PlayReady HW patch package in the Android SDK root directory.

Step 4 Run **./pack_install.sh ${ANDROID_BUILD_TOP}** in the Android SDK root directory to incorporate the patch content to the SDK.

Step 5 Update the library file, configuration file, and Provision-related files in the PlayReady release package obtained from Discretix to the SDK by referring to description in the **device/hisilicon/bigfish/prebuilts/drm/readme_cn.txt** file.

- Update the PlayReady library file.

  Copy **/Software/SW/*.so** in the PlayReady HW release package obtained from Discretix to the **device/hisilicon/bigfish/prebuilts/drm** directory of the SDK.

- Update the PlayReady TA.

  Copy the **/Software/Service/obj/EXECUTABLES/task_QA_TZService_intermediates/task_QA_TZService.elf** file in the PlayReady release package obtained from Discretix to the **device/hisilicon/bigfish/sdk/source/plugin/tvp/trustedcore/release/internal_tasks** directory of the SDK.

- Update the Provision tool.
  - Copy following files in the **/Provisioning/SecuredProvisioning** directory of the PlayReady release package obtained from Discretix to the **device/hisilicon/bigfish/prebuilts/drm/Provisioning** directory of the SDK.
    - cek.bin
    - cekVerifier.bin
    - DxDrmConfig.txt
    - DxPrdyPkgEnc
  - Copy the PlayReady release package obtained from Discretix.

    Copy the **/Provisioning/SecuredProvisioning/provTest/libs/armeabi/securedProvTest** file to the **device/hisilicon/bigfish/prebuilts/drm/Provisioning** directory of the SDK.

----End

2.3 Compilation

The HW version depends on the secure chipset supporting the TrustZone feature and the SDK. Using Hi3719C V100 as an example, run the following commands in the SDK root directory in sequence:

- source build/envsetup.sh
- lunch Hi3719CV100-eng or lunch Hi3719CV100-user
- ./device/hisilicon/Hi3719CV100/build/tvp.sh (After this command is run, the security configuration of the SDK is enabled.)
- make bigfish -j32 2>&1 | tee bigfish.log

2.4 Signature

Make signatures for the **boot** and **trustedcore** (trustedcore.img) images by referring to operations in the *TrustZone Development Guide*.

2.5 Burning

Burn the secure **boot**, **trustedcore**, and other images using the HiTool by referring to operations in the *TrustZone Development Guide*.

2.6 Test Description

Test the test cases provided by the PlayReady official test website using **PlayReady Demo APK** by referring to description in the **device/hisilicon/bigfish/prebuilts/drm/readme_cn.txt** file.

If encrypted files need to be played, the set top box (STB) must access the Microsoft license server.

2.6.1 Test Preparations

- Check whether the STB has the following files:
  - /system/lib/libDxDrmServer.so
  - /system/lib/libdxprdyDrmOem.so
  - /system/lib/drm/libDxPrRecommended.so
  - DrmAssist-Recommended.apk in the /system/app or /data/app directory
  - /system/etc/DxDrmConfig.txt or /data/DxDrm/DxDrmConfig.txt
  - /data/DxDrm/cek.bin
  - /data/DxDrm/cekVerifier.bin
  - /data/DxDrm/DxPrdyPkgEnc
  - /data/DxDrm/securedProvTest
  - /data/DxDrm/run_secure_prov.sh
- Check whether the value of **drm.service.enabled** in the **/system/build.prop** file is **true**.

- Check whether **drmserver** is running.
- Pay attention to following notes when configuring files:
  - The default directory of the **DxDrmConfig.txt** file is **/system/etc/**. You can put the file in the **/data/DxDrm** directory. You are advised to put the file in the default directory to prevent it from being deleted by mistake.
  - In the **DxDrmConfig.txt** file, the location of the **sfs** secure file system is configured as follows:

    SfsLocation = /data/sfs

    The **/data/sfs** directory is used only for debugging. The default directory of the secure file system is **/drmsfs**. You are advised to mount an independent flash memory to the **/drmsfs** directory to prevent the secure file system from being deleted by mistake.

    When **SfsLocation** is changed, the corresponding path in the **run_secure_prov.sh** file must be changed accordingly.

2.6.2 Certificate Initialization

Before development debugging, the certificate needs to be manually initialized when a board is powered on. The detailed procedure is as follows:

Step 1 Run **cd /data/DxDrm**.

Step 2 Run **chmod 755 run_secure_prov.sh**.

Step 3 Run **./run_secure_prov.sh**.

Step 4 Check whether any initialization failure is prompted in the log.

Step 5 Check whether the **/data/sfs/dxprdy/dxprdy/dxprdy.sfs** file exists and its attribute is **666**.

----End

CAUTION

- During mass production in a factory, a dedicated Provision process is provided for executing certificate burning and initialization. Therefore, no manual certificate initialization is required.
- If data in the **/data/DxDrm** directory is not corrupted, certificate initialization only needs to be executed once.

2.6.3 Playing Smoothstreaming Files Using a Web Page

To play a smoothstreaming file using a web page, perform the following steps:

Step 1 Run **Playready Demo APK** (**DrmAssist-Recommended.apk**) whose name and icon both are **DRM**.

Step 2 Click **UrlPlay** and enter the following test page for smoothstreaming provided by Microsoft:

http://playready.directtaps.net/smoothstreaming/

Step 3 Select an arbitrary manifest file and play the file.

----End

2.6.4 Locally Playing an ASF File

To locally play an ASF file, perform the following steps:

Step 1 Download a test stream to a local PC from the PlayReady server provided by Microsoft.

Open the http://playready.directtaps.net/ page, click **PYV/PYA**, and download the corresponding test cases and stream provided by the PlayReady official test website to the local PC.

Step 2 Run **Playready Demo APK** (**DrmAssist-Recommended.apk**) whose name and icon both are **DRM**.

Step 3 Click **LocalPlay**, enter the directory of the file for testing, and select the file for playing.

Step 4 Check whether the rights for playing the file are valid using **CheckRights**. If not, click **GetRightsByPlugin** or **GetRightsByApplication** to obtain the corresponding rights.

Step 5 Click **Play** to play the file.

----End

2.7 Factory Manufacturing

The HW version requires the secure startup and TrustZone features. During mass production in a factory, the following procedure needs to be performed:

- Burn the HDCP root key, encrypt the HDCP key, and save it to a flash memory for protection during HDMI output.
- Burn the RSA root public key for verifying the **Key Area** data in the **boot** image when the chipset starts.
- Burn the **PlayReady SFS Key** for protecting the system data of the PlayReady security files during encryption and decryption.
- Burn and initialize the PlayReady certificate.
- Enable secure startup and TrustZone.

All codes for burning preceding data can be found in the **device/hisilicon/bigfish/sdk/sample/advca/sample_factory_drm.c** file on the SDK.

2.7.1 Data List of the OTP Chipset

Table 2-1 lists data to be burnt to OTP chipset as required by the TVP PlayReady scheme.

Table 2-1 Data list of the OTP chipset

Name	Position	Size	Description
RSA_Root Key	0x2f0–0x3 ff	E: 16 bytes N: 256 bytes	The RSA Root Public key is used to verify **RSA_Ext_Key** of **key_area** in the **boot** image, and cannot be read once being locked.
HDCP_Roo tKey	0xc0–0xcf	16 bytes	The HDCP root key is used to encrypt and decrypt keys and cannot be read once being locked.

Name	Position	Size	Description
pr_sfs_key	0x130–0x 13f	16 bytes	This key is used to encrypt and decrypt the PlayReady certificate and other PlayReady data saved locally. This key is located in the trust zone of the OTP chipset and can be accessed only by a secure CPU when **otp_tz_area_enable** is enabled.
otp_tz_area _enable	0x4[2]	1 bit	Enable flag of the TrustZone function of the OTP chipset. After this flag is enabled, a secure CPU can access data in the trust zone of the OTP chipset.
soc_tz_enab le	0x4[1]	1 bit	Enable flag of the TrustZone function of the system on chip (SOC) chipset.
SCS_activat ion	0x18[0]	1 bit	Enable flag of secure startup.

2.7.2 Preparing Data

2.7.2.1 RSA Root Public Key

Update the **rsa_root_key_pub[]** array in the **sample_factory_drm.c** file using the array in the **root_rsa_pub.h** file of the RSA root key generated when a signature is made for the boot image in section 2.4 "Signature."

2.7.2.2 OEM Seed Key

The **oem_seed_key[]** array in the **sample_factory_drm.c** file contains seed keys for various encryption and decryption keys, which are defined by OEMs. The seed keys can be used to generate the **HDCP_RootKey**, **R2R_RootKey**, and **pr_sfs_key** based on the OEM requirements.

Based on the definition of **HI_FAC_KEY_MODE_E**, keys can be generated in following modes:

- HI_FAC_KEY_MODE_UNIFIED

  All STB chipsets use the same encryption and decryption key.
- HI_FAC_KEY_MODE_UNIQUE

  Each STB chipset uses an independent encryption and decryption key.

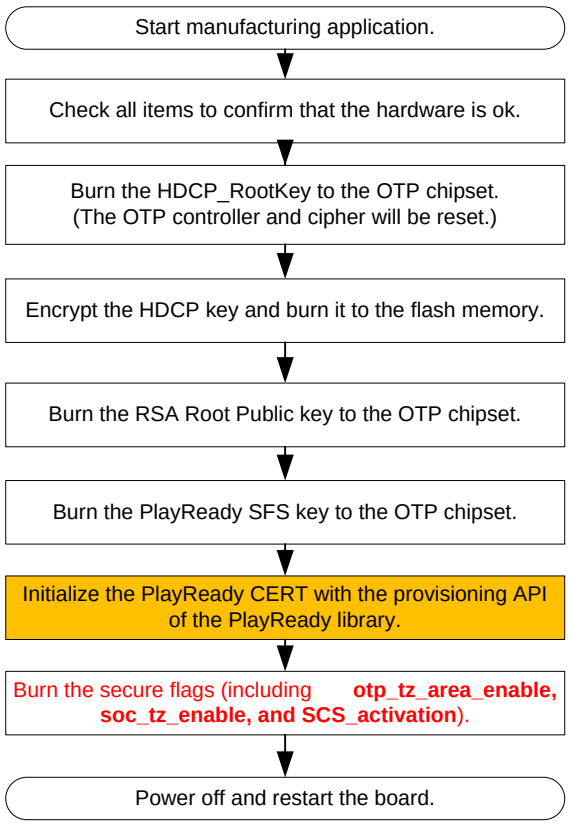
  Based on a fixed algorithm, seed key, type of the encryption and decryption key, and chip ID, the corresponding encryption and decryption key can be calculated.
- HI_FAC_KEY_MODE_RANDOM

  Each STB chipset uses a random encryption and decryption key.

2.7.3 Data Burning Procedure

Figure 2-2 shows the factory manufacturing process for the PlayReady in HW version.

Figure 2-2 Factory manufacturing process for the PlayReady in HW version



The step marked orange is to burn and initialize the certificate. For details, see section 2.7.5 "PlayReady Certificate Burning."

2.7.4 APIs for Factory Manufacturing

The **device/hisilicon/bigfish/sdk/sample/advca/sample_factory_drm.c** file on the SDK is a sample file for manufacturing the PlayReady HW. This file provides the following interfaces for burning data to the OTP chipset.

- HI_FAC_Set_HDCP_RootKey: used to burn the HDCP root key.
- HI_FAC_Encrypt_HdcpKey: used to encrypt the HDCP key.
- HI_FAC_Set_RSA_RootKey: used to burn the RSA Root Public key.
- HI_FAC_Set_PR_SFS_Key: used to burn the encryption and decryption key for the PlayReady security file system.
- HI_FAC_Set_Secure_Flag: used to burn the otp_tz_area_enable, soc_tz_enable, and SCS_activation flags and enable the TrustZone and secure startup functions.

2.7.5 PlayReady Certificate Burning

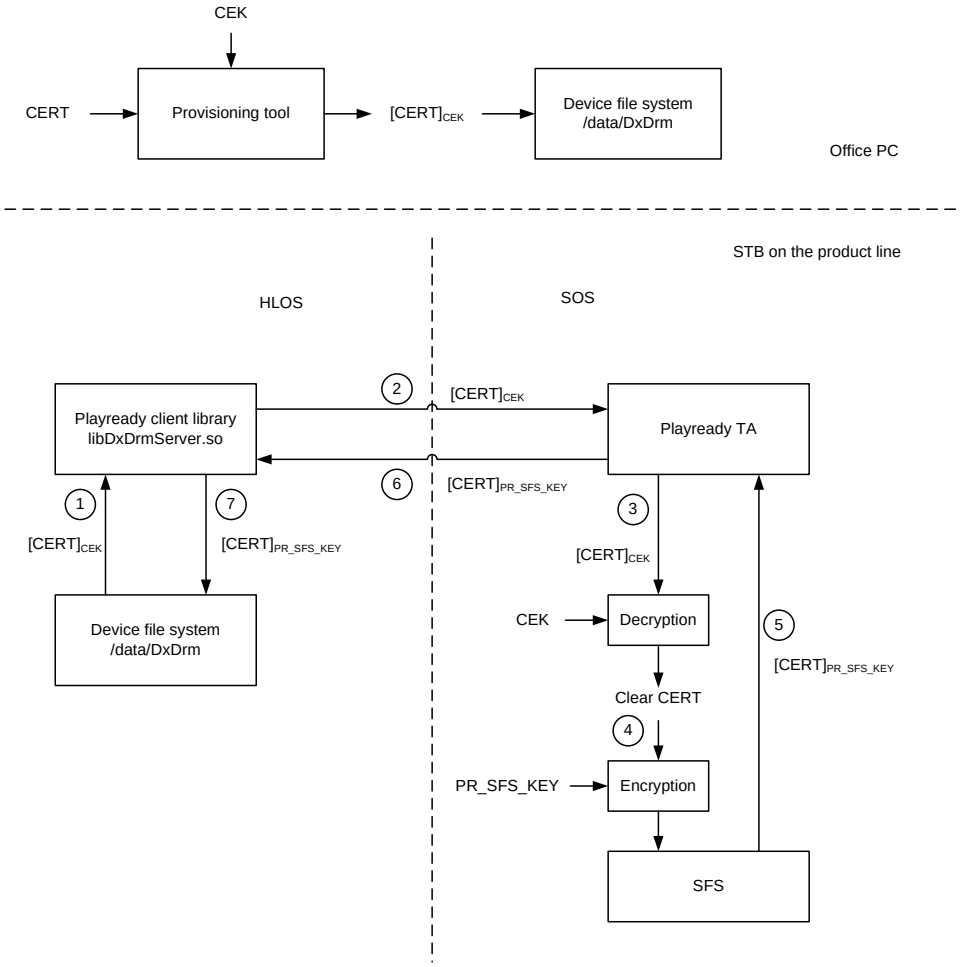
2.7.5.1 PlayReady Certificate Burning Scheme

OEMs use the Provisioning tool provided by Discretix to encrypt the PlayReady Model certificate (CERT for short) with the encryption key customer encryption key (CEK), and then put the encrypted certificate to the STB file system.

During STB testing in the factory, the **Provisioning API** provided by **libDxDrmServer.so** is executed to decrypt the certificate. Then, the certificate is encrypted again using the **PR_SFS_KEY** burnt to the OTP chipset.

Figure 2-3 shows the PlayReady certificate burning scheme.

Figure 2-3 PlayReady certificate burning scheme



HLOS: non-secure file system Linux

SOS: secure file system

2.7.5.2 PlayReady Certificate Burning Procedure

To burn the PlayReady certificate, do as follows:

Step 1 Use the Provisioning tool provided by Discretix to encrypt the PlayReady model certificate CERT on the local PC. The used encryption key is CEK.

Step 2 Put the encrypted certificate **[CERT]$_{CEK}$** to the **/data/DxDrm** directory of the STB.

Step 3 Execute **Provisioning API SetProvisioningCEK()** provided by **libDxDrmServer.so** to set the decryption key CEK of **[CERT]$_{CEK}$**.

Step 4 Execute **Provisioning API VerifyStoredCEK ()** provided by **libDxDrmServer.so** to confirm the setting of the decryption key CEK of [CERT]$_{CEK}$.

Step 5 Execute **Provisioning API StartSecuredProvisioning ()** provided by **libDxDrmServer.so** to finish certificate burning.

The PlayReady library will send [CERT]$_{CEK}$ to the secure memory for decryption, encrypt the PR_SFS_KEY in the secure domain of the OTP chipset, send [CERT]$_{PR\_SFS\_KEY}$ to the non-secure side, and save it to the STB file system. For details, see Figure 2-3.

Step 6 Execute **Provisioning API VerifySecuredProvisioning ()** provided by **libDxDrmServer.so** to confirm whether certificate burning succeeds.

----End

2.7.5.3 APIs for PlayReady Certificate Burning

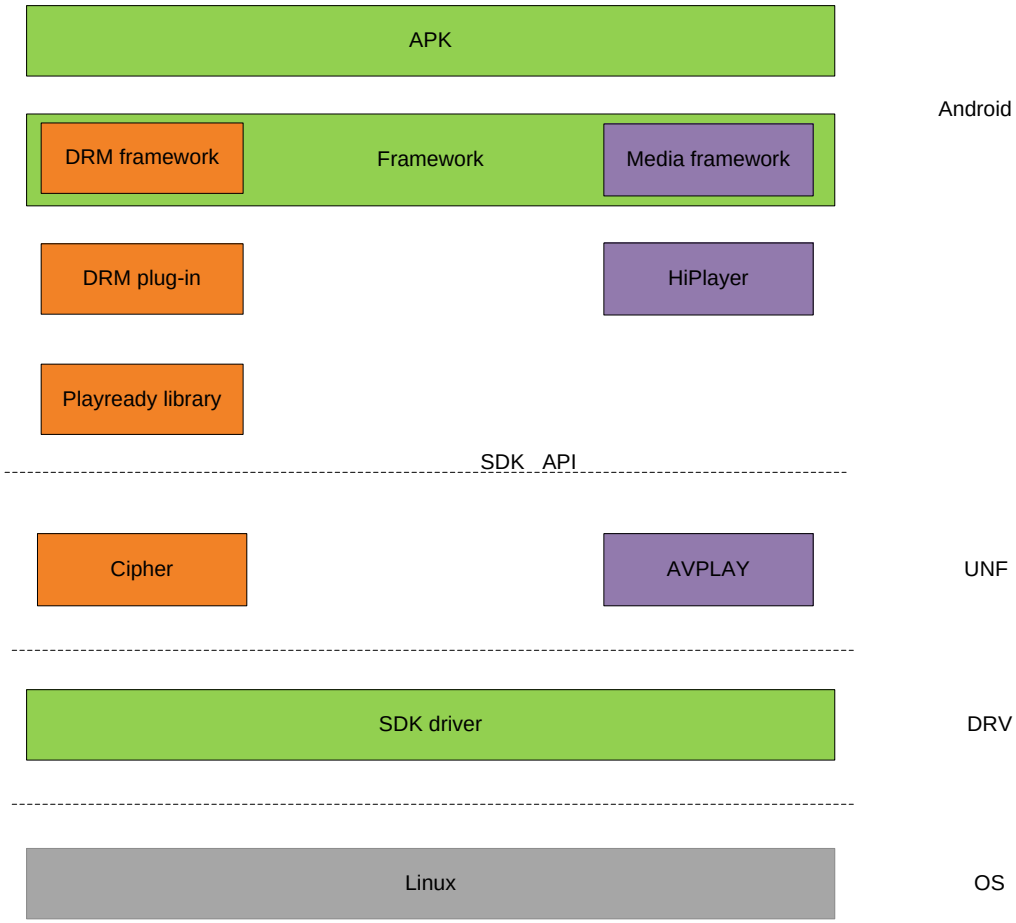
See the *Secured Provisioning Design* in the PlayReady release package provided by Discretix.

3 PlayReady SW Development

3.1 Block Diagram of the Scheme

Figure 3-1 shows the block diagram of the PlayReady SW scheme.

Figure 3-1 Block diagram of the PlayReady SW scheme



3.2 Preparations

- Apply for a PlayReady company certificate from Microsoft and use it to generate a model certificate, which includes the following files:
  - bgroupcert.dat
  - zgpriv.dat
  - devcerttemplate.dat
  - priv.dat
- Obtain a PlayReady SW patch package from HiSilicon, which contains the following library files:
  - libhi_playready.so
  - libhi_playready_protect.so
  - libplayreadyplugin.so

3.3 Compilation Instruction

Take Hi3719C V100 as an example:

- Do not compile the HW version.

  Do not run the **device\hisilicon\Hi3719C V100\build\tvp.sh** script and ensure that the value of **HISI_TRUST_VIDEO_PATH** in the **device\hisilicon\Hi3719C V100\BoardConfig.mk** file is **false**.
- Ensure that **PRODUCT_PROPERTY_FOR_DRM** in the **device\hisilicon\Hi3719C V100\device.mk** file is **true**.

3.3.1 Compiling Classic PlayReady

Step 1 Input **mm** in **device/hisilicon/bigfish/sdk/source/component/playready** and **libhi_playready_protect.so**, **libhi_playready.so**, and **libplayreadyplugin.so** are obtained after compilation.

Step 2 Push the libraries **libhi_playready_protect.so** and **libhi_playready.so** to **system/lib/** and push **libplayreadyplugin.so** to **/system/lib/drm/**.

----End

3.3.2 Compiling Modular PlayReady

Step 1 Compile classic PlayReady by following the steps stated in section 3.3.1 "Compiling Classic PlayReady."

Step 2 Input **mm** in the directory of **/device/hisilicon/bigfish/hidolphin/component/drm/source/playready/module_drmplugin_playready** and obtain **libplayreadydrmplugin.so**.

Push **libplayreadydrmplugin.so** to the directory of **/system/vendor/lib/mediadrm**.

----End

3.4 Test Description

3.4.1 Configuration File Description

You can use the **playreadyconfig.txt** configuration file to configure the running environment of the PlayReady SW version:

- The default path of the **playreadyconfig.txt** file on a board is **/system/etc** or **/data/playready**.
- The following is an example showing contents of the **playreadyconfig.txt** file. You can change the value after the equal mark (=). Do not leave space before or after the equal mark (=).

```
CertPath=/data/playready/prpd/         /*Path of the certificate*/
LocalStorePath=/data/playready/prdata.localstore  /*Path of temporary
data*/
KeyFilePath=/data/playready/prpd/      /*Path of the device
certificate*/
CertProtection=enable      /*Whether the certificate is encrypted (the
valid value is disable and enable)*/
CertRootKeyType=stb_root_key       /*Type of the key for encrypting the
certificate*/
LogLevel=error         /*Log output level (no, error, info)*/
```

- If no such configuration file is available, or the configuration items are invalid, use the following default configuration and store the certificate in plain text. This certificate is only used for development debugging.

```
CertPath=/data/playready/prpd/
LocalStorePath=/data/playready/prdata.localstore
KeyFilePath=/data/playready/prpd/
CertProtection=disable
LogLevel=error
```

- If the certificate needs to be encrypted for storage, set the contents of the **playreadyconfig.txt** file as follows:

```
CertProtection=enable
CertRootKeyType=stb_root_key
```

- When the certificate needs to be encrypted for storage, the key needs to burnt for encrypting the certificate. For details about key burning and certificate encryption, see section 3.5 "Factory Manufacturing."
- The CertRootKey for encrypting the certificate is stored in the OTP chipset and cannot be read once being locked.

3.4.2 Test Preparations

Step 1 Copy the library files in the PlayReady SW patch package obtained from HiSilicon to the following directories:

- /system/lib/libhi_playready.so
- /system/lib/libhi_playready_protect.so

- /system/lib/drm/libplayreadyplugin.so
- /system/vendor/lib/mediadrm/libplayreadydrmplugin.so

Step 2 Configure the running environment of the PlayReady SW version by referring to operations in section 3.4.1 "Configuration File Description."

Step 3 Create a directory **/data/playready/prpd** on the board and ensure that **/data/playready** and its child directories can be read and written. Then, copy the certificate file to the **/data/playready/prpd** directory. If the certificate needs to be encrypted, refer to operations in section 3.5 "Factory Manufacturing."

Step 4 Ensure that the /system/app/DrmAssist-Recommended.apk file exists.

Step 5 Check whether the values of drm.service.enabled and drm.client.enabled in the /system/build.prop file are true.

Step 6 Check whether **drmserver** is running.

Step 7 If the system has enabled SELinux, decompress the release package of the project code. Place the library files of the PlayReady patch package in the four directories (under **/out/target/system**) specified in Step 1. Then recompile the release package and reburn the image to enable the proper function of the Android PlayReady. In addition, restart the system after Step 2 and Step 3 are complete.

----End

3.4.3 Playing Smoothstreaming Files Using a Web Page

To play a smoothstreaming file using a web page, perform the following steps:

Step 1 Push the library **libhi_smoothstreaming.so** to **/system/lib.** Note that the size of the library should be greater than 800 KB, and if it is about 5 KB, compile the library in **/device/hisilicon/bigfish/hidolphin/component/smoothstreaming.**

Step 2 Run **Playready Demo APK** (**DrmAssist-Recommended.apk**) whose both name and icon are **DRM**.

Step 3 Click **UrlPlay** and enter the following test page for smoothstreaming provided by Microsoft:

http://playready.directtaps.net/smoothstreaming/

Step 4 Select an arbitrary manifest file and play the file.

----End

3.4.4 Locally Playing an ASF File

To locally play an ASF file, perform the following steps:

Step 1 Download a test stream to a local PC from the PlayReady server provided by Microsoft.

Open the **http://playready.directtaps.net/** page, click **PYV/PYA**, and download the corresponding test cases and stream provided by the PlayReady official test website to the local PC.

Step 2 Run **Playready Demo APK** (**DrmAssist-Recommended.apk**) whose name and icon both are **DRM**.

Step 3 Click **LocalPlay**, enter the directory of the file for testing, and select the file for playing.

Step 4 Check whether the rights for playing the file are valid using **CheckRights**. If not, click **GetRightsByPlugin** or **GetRightsByApplication** to obtain the corresponding rights.

Step 5 Click **Play** to play the file.

----End

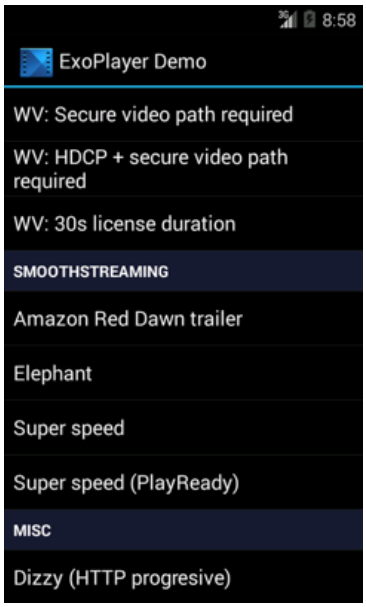
3.4.5 Exoplayer Test

Step 1 Compile modular PlayReady by following the steps stated in section 3.3.2 "Compiling Modular PlayReady."

Step 2 Create a **/data/mediadrm/**playready directory on the board. Ensure that the files and directories in this directory have read and write properties. Then copy four certificates to this directory.

Step 3 Open Exoplayer, select **Super speed** to test the network connection and whether the media CODEC works properly. Select **Super speed (PlayReady)** to test whether the modular PlayReady works properly.

Figure 3-2 Selecting Super speed (PlayReady)



----End

3.5 Factory Manufacturing

The PlayReady SW version is applicable to scenarios where the certificate needs to be encrypted for storage. During mass production in a factory, the following procedure needs to be performed:

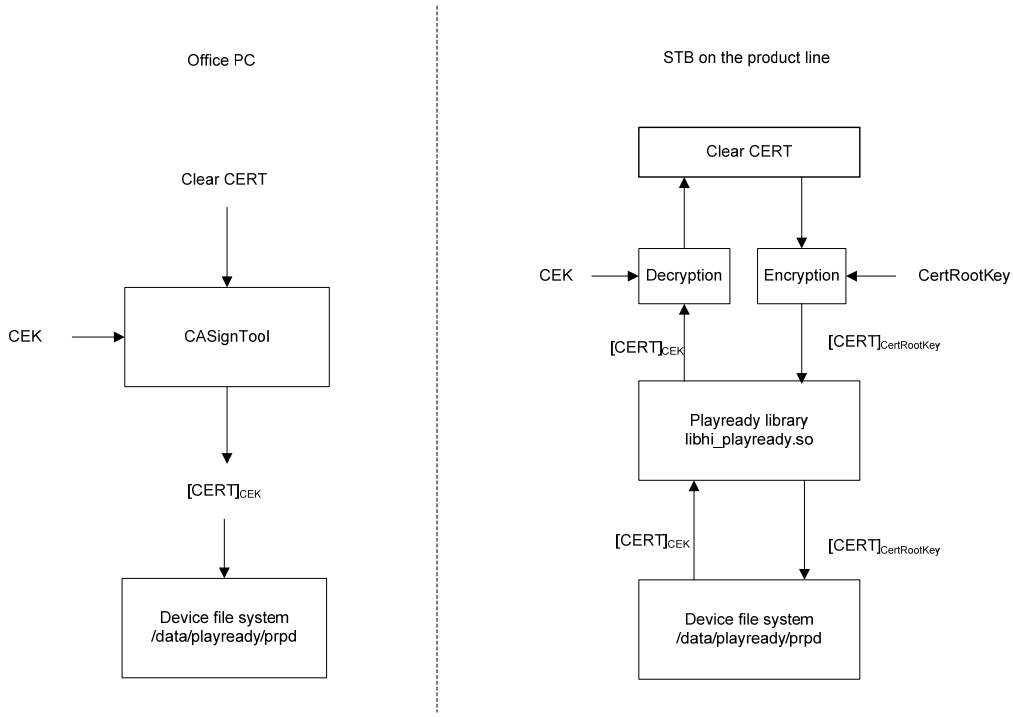
- Encrypt the license to ensure secure transmission from OEMs to the factory.
- Burn the CertRootKey for encrypting the certificate to the OTP chipset.

- Encrypt the certificate using the CertRootKey and save the certificate to the board flash memory.

3.5.1 Factory Manufacturing Scheme

Figure 3-3 shows the factory manufacturing process.

Figure 3-3 Factory manufacturing process



NOTE

You can also directly encrypt the plaintext certificate by using CertRootKey on the board. That is, step 3 in the preceding figure can be skipped and performed separately.

3.5.2 Factory Manufacturing Procedure

The factory manufacturing procedure is as follows:

Step 1 Use the CASignTool (version 2.15 or later) to encrypt the PlayReady device certificate (including bgroupcert.dat, zgpriv.dat, devcerttemplate.dat, and priv.dat) generated in section 1.5 "Applying for and Using a PlayReady Certificate" on the local PC.

The encryption algorithm is AES128-CTR and the used key is the 16-byte CEK generated on the customer side.

Step 2 Copy the CEK to the board's PlayReady certificate directory during production line testing in a factory.

Step 3 Burn the CertRootKey to the OTP chipset.

Step 4 Execute the certificate initialization function, decrypt the certificate using the CEK to the memory, and encrypt the certificate using the CertRootKey to the flash memory.

----End

3.5.3 APIs for Factory Manufacturing

The **libhi_playready.so** file provides the following API functions for factory manufacturing:

- **HI_PLAYREADY_Provision_IsCertRootKeyWrited**: used to check whether the CertRootKey encryption key has been burnt.
- **HI_PLAYREADY_Provision_SetCertRootKey**: used to burn the CertRootKey for encrypting the certificate to the OTP chipset.
- **HI_PLAYREADY_Provision_CertInit**: used to initialize the certificate. Decrypt the certificate using the CEK and then encrypt the certificate using the CertRootKey in the OTP chipset.

3.5.4 Description of APIs for Factory Manufacturing

HI_PLAYREADY_Provision_IsCertRootKeyWrited

[Description]

Check whether the CertRootKey encryption key has been burnt.

[Syntax]

```
HI_S32 HI_PLAYREADY_Provision_IsCertRootKeyWrited(HI_BOOL *bWrited);
```

[Parameter]

Parameter	Description	Input/Output
bWrited	Indicates whether the CertRootKey encryption key has been burnt.	Output

[Return Value]

Return Value	Description
HI_SUCCESS	The CertRootKey is burnt successfully.
Others	The CertRootKey burning fails.

[Requirement]

Library: libhi_playready.so

[Note]

None

[Example]

None

[See Also]

HI_PLAYREADY_Provision_SetCertRootKey

HI_PLAYREADY_Provision_SetCertRootKey

[Description]

Burn the 16-byte CertRootKey to the OTP chipset.

[Syntax]

```
HI_S32 HI_PLAYREADY_Provision_SetCertRootKey(HI_U8 *pu8CertRootKey);
```

[Parameter]

Parameter	Description	Input/Output
pu8CertRootKey	16-byte CertRootKey. If the parameter value is a null pointer, burn an arbitrary value to the OTP chipset.	Input

[Return Value]

Return Value	Description
HI_SUCCESS	The CertRootKey is burnt successfully.
Others	The CertRootKey burning fails.

[Requirement]

Library: libhi_playready.so

[Note]

The CertRootKey will be locked after being burnt, and cannot be read.

Unless otherwise required, you are advised to burn an arbitrary value to the OTP chipset to enhance certificate and key security.

CAUTION

After CertRootKey is burnt, ensure that the dynamically generated device certificate **/data/playready/prpd/keyfile.dat** and license **/data/playready/prdata.localstore** are deleted. Otherwise, the PlayReady library fails to run due to the mismatched device certificate key.

[Example]

None

[See Also]

HI_PLAYREADY_Provision_IsCertRootKeyWrited

HI_PLAYREADY_Provision_CertInit

[Description]

Initialize the certificate. Decrypt the certificate using the CEK and then encrypt the certificate using the CertRootKey in the OTP chipset.

[Syntax]

```
HI_S32 HI_PLAYREADY_Provision_CertInit(HI_U8 *pu8CEK);
```

[Parameter]

Parameter	Description	Input/Output
pu8CEK	16-byte CEK for encrypting the certificate. If a null pointer is provided, the certificate is not encrypted using the CEK. You can encrypt the plaintext certificate directly.	Input

[Return Value]

Return Value	Description
HI_SUCCESS	The certificate is initialized successfully.
Others	The certificate initialization fails.

[Requirement]

Library: libhi_playready.so

[Note]

The certificate encrypted using the CEK needs to be stored in the path described in section 3.4.1 "Configuration File Description."

[Example]

None

[See Also]

None

4 Application Development

4.1 Introduction to PlayReady Demo APK

The PlayReady Demo APK provides the functions of locally playing an ASF file and playing a smoothstreaming file using a web page, and supports acquisition of stream rights.

The PlayReady Demo APK is implemented based on the Android DRM framework and standard interface at the media framework Java layer. The right acquisition is implemented based on the **DrmManagerClient.java** file.

The source code of the PlayReady Demo APK is stored in the **device/hisilicon/bigfish/development/apps/PlayReadyDemo** directory on the Android SDK provided by HiSilicon. The compiled APK is named **DrmAssist-Recommended.apk**. The icon and name are both **DRM**.

4.2 Drm Header Acquisition

The **Drm Header** data is used to acquire rights for encrypting and playing streams. The **Drm Header** of PlayReady corresponds to the **Rights Management Header** in **PlayReady Header Objects**. To obtain the standards of PlayReady header objects, access the following address to download this file:

http://www.microsoft.com/playready/documents/

The PlayReady plug-in can parse the **Drm Header** data from the Manifest files of a local ASF file or an online smoothstreaming file.

The APK can call the **acquireDrmInfo()** interface of **DrmManagerClient** and obtain the **Drm Header** data in the format of character strings using the PlayReady plug-in to acquire rights for encrypting and playing streams. For the reference code, see the **WebActivity.java** or **AcquireRightsByApp.java** file in **PlayReady Demo APK**.

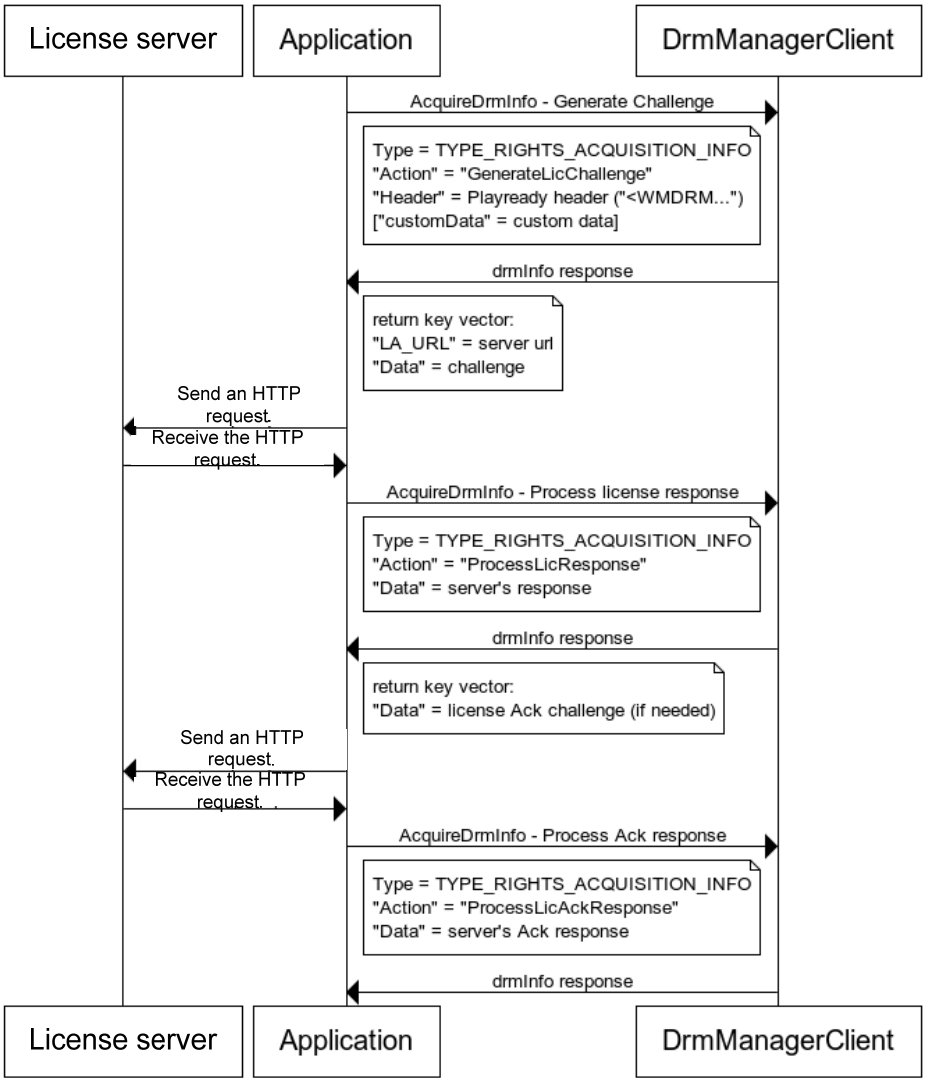
CAUTION

The PlayReady plug-in cannot parse contents of online files. Therefore, you need to download the manifest file of the online smoothstreaming file to the local PC and save it as a ***.ismc** file. Use the complete path of the file (including the file name) as the input parameter for obtaining the DrmHeader data.

4.3 Right Acquisition Process

Figure 4-1, Figure 4-2, Figure 4-3, and Figure 4-4 show the license acquisition, domain joining, domain leaving, and meter processes, respectively.

Figure 4-1 License acquisition process (server communication by application)



For the right acquisition process code, see the **WebActivity.java** or **AcquireRightsByApp.java** file in **PlayReady Demo APK**.

Figure 4-2 Process for joining a domain

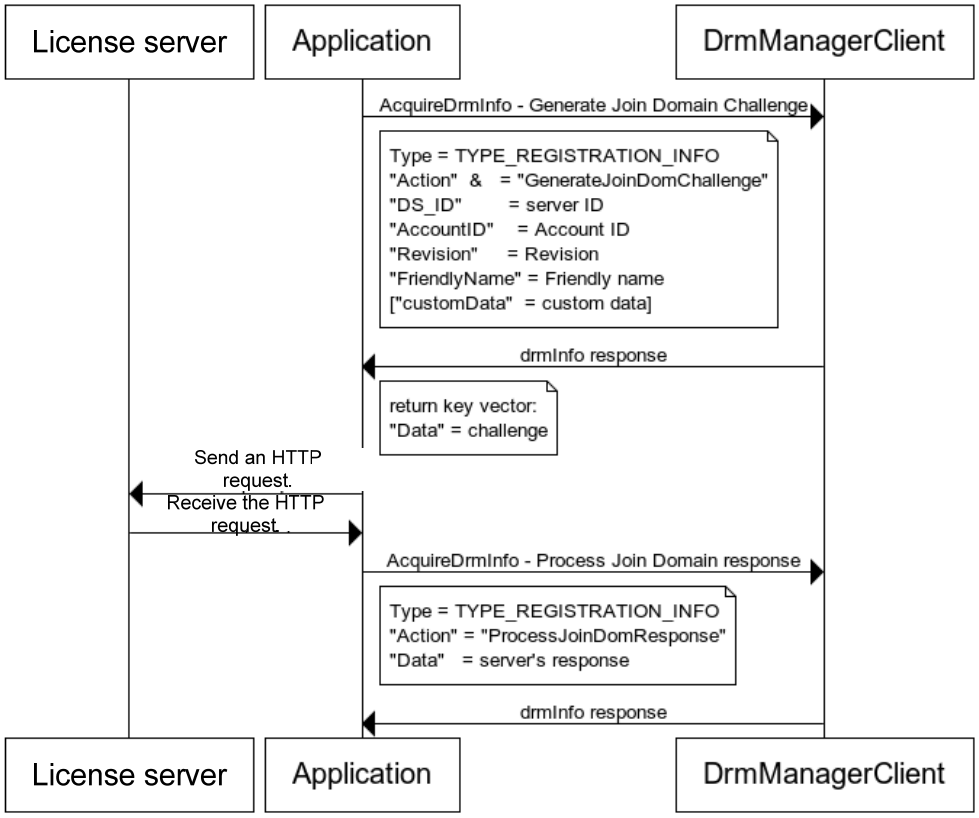


Figure 4-3 Process for leaving a domain

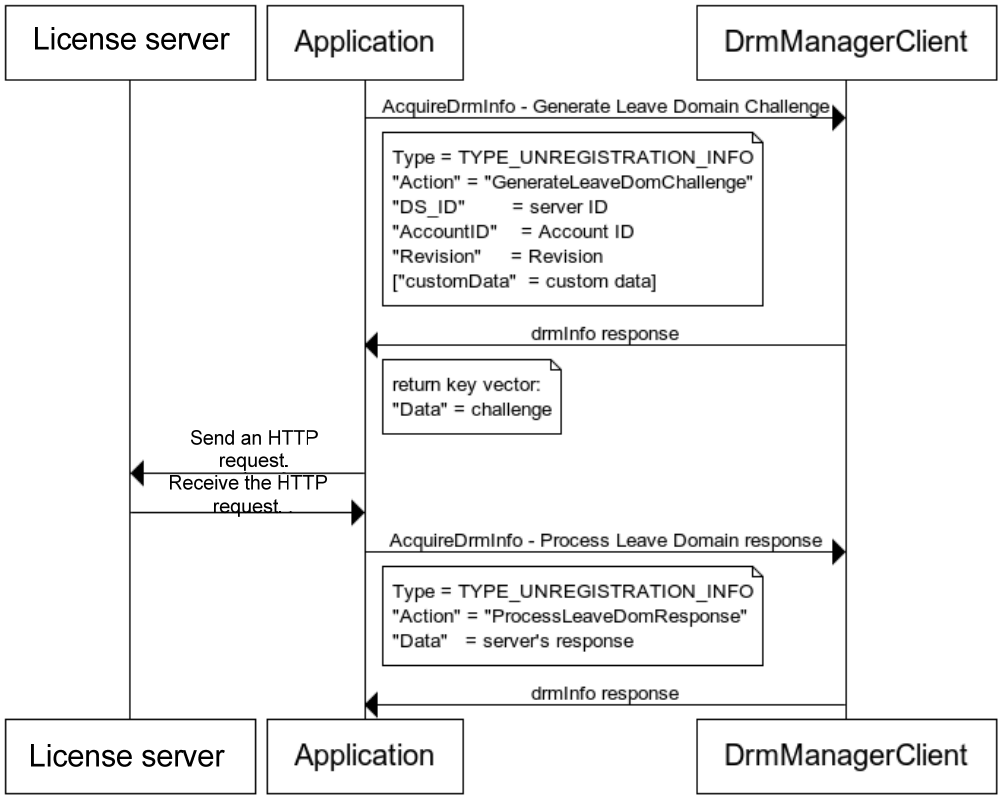
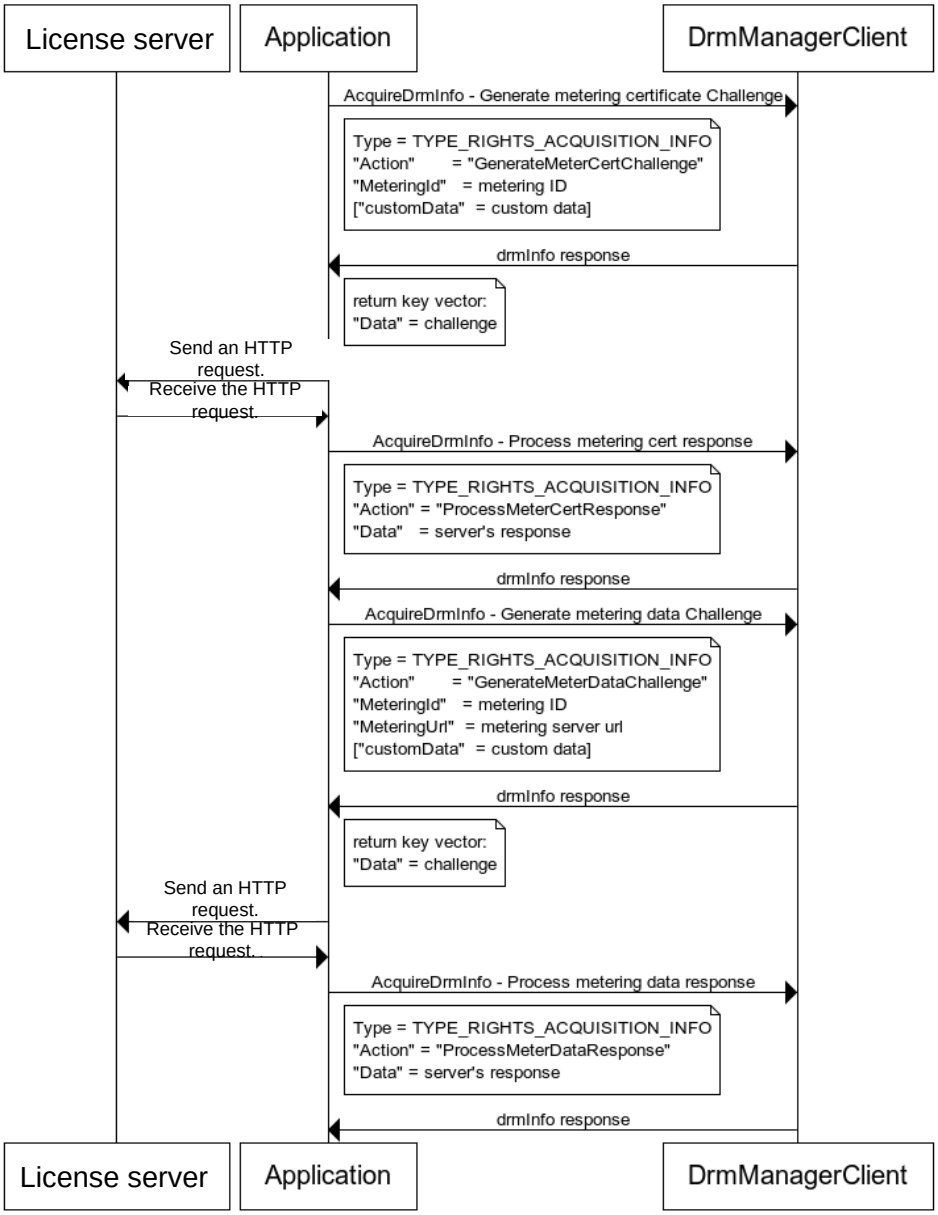


Figure 4-4 Metering process



4.4 Media Decryption and Playing Process

A media player calls the C++ interface at the Drm framework layer to decrypt and play encrypted PlayReady streams. For the corresponding API, see the *DrmManagerClient.h*.

4.4.1 Local File Decryption Process

To decrypt a local file, do as follows:

Step 1 Open a decryption session.

```
sp<DecryptHandle> openDecryptSession(int fd, off64_t offset, off64_t length,
```

```
const char* mime);
```

or

```
sp<DecryptHandle> openDecryptSession(const char* uri, const char* mime);
```

Step 2 Consume rights.

```
status_t consumeRights(sp<DecryptHandle> &decryptHandle, int action, bool
reserve);
```

This interface can be called for each decryption session only once.

Step 3 Repeatedly call the following two interfaces to read and decrypt data until the player stops.

```
mOriginalMediaSource->read(MediaBuffer **buffer, const ReadOptions
*options)
```

and

```
status_t decrypt(sp<DecryptHandle> &decryptHandle, int decryptUnitId, const
DrmBuffer* encBuffer, DrmBuffer** decBuffer, DrmBuffer* IV = NULL);
```

Step 4 Close the decryption session when the player stops.

```
status_t closeDecryptSession(sp<DecryptHandle> &decryptHandle);
```

----End

4.4.2 Web Stream Decryption Process

To decrypt a web stream, do as follows:

Step 1 Open a decryption session.

```
sp<DecryptHandle> openDecryptSession(const char* uri, const char* mime);
```

CAUTION

When playing an online smoothstreaming file using a web page, ensure that the **uri** is a fixed string "**ms-smooth-streaming**."

Step 2 Set the **DrmHeader**.

```
status_t initializeDecryptUnit(sp<DecryptHandle> &decryptHandle, int
decryptUnitId, const DrmBuffer* headerInfo);
```

Note that the **headerInfo** parameter contains whole contents of **PlayReady Header Objects**.

Step 3 Consume rights.

```
status_t consumeRights(sp<DecryptHandle> &decryptHandle, int action, bool
reserve);
```

This interface can be called for each decryption session only once.

Step 4 Repeatedly call the following two interfaces to read and decrypt data until the player stops.

```
mOriginalMediaSource->read(MediaBuffer **buffer, const ReadOptions
*options)
```

and

```
status_t decrypt(sp<DecryptHandle> &decryptHandle, int decryptUnitId, const
DrmBuffer* encBuffer, DrmBuffer** decBuffer, DrmBuffer* IV = NULL);
```

Step 5 Close the decryption session when the player stops.

```
status_t finalizeDecryptUnit(sp<DecryptHandle> &decryptHandle, int
decryptUnitId);
```

and

```
status_t closeDecryptSession(sp<DecryptHandle> &decryptHandle);
```

----End

4.5 DrmManagerClient APIs

4.5.1 API Fields

This document describes the reference information about APIs with nine API fields. Table 4-1 describes the nine API fields.

Table 4-1 lists various API fields.

Table 4-1 API fields

Field	Function
Purpose	Describes the major function of an API.
Syntax	Lists the syntax of an API.
Description	Describes the work process of an API.
Parameter	Describes the parameters and attributes of an API.
Return Value	Describes the return values of an API.
Error Code	Describes the error codes of an API.
Requirement	Lists the header files and library files required when an API is called.
Note	Describes the notes for calling an API.
Example	Provides the examples of calling an API.

4.5.2 Data Structure Fields

This section describes data structures with five reference fields. Table 4-2 describes the five reference fields.

Table 4-2 Data structure fields

Field	Function
Purpose	Describes the function of a data structure.
Syntax	Describes the definition of a data structure.
Parameter	Describes the parameters of a data structure.
Description	Describes the usage of a data structure and the usage notes.
Reference	Describes the reference information.

4.5.3 API Summary

4.5.3.1 License Acquisition Related APIs in DrmManagerClient.java

- getMetadata: used to obtain the metadata of the content.
- getOriginalMimeType: used to obtain the MIME type of the original content.
- canHandle: used to check whether the MIME type or content is supported.
- acquireRights: used to execute the full right acquisition process, including communication with the license server.
- acquireDrmInfo: used to return registration, deregistration, and right request information.
- processDrmInfo: used to process specified DRM information such as the right request and initiator. The PlayReady plug-in does not support this interface.
- getConstraints: used to return right constraint information of the content.
- setOnInfoListener: used to register a callback function for listening information. With this interface, when the DRM plug-in is registered or sending a right request, the status or warning information can be returned.
- setOnEventListener: used to register a callback function for listening events. With this interface, the DRM processing information can be returned by the DRM framework.
- setOnErrorListener: used to register a callback function for listening errors. With this interface, error information can be returned by the DRM framework.
- getDrmObjectType: used to return the right protection type of the content or MIME type. The PlayReady plug-in does not support this interface.
- checkRightsStatus: used to check whether the right for reading the content is valid.
- removeRights: right for removing the content.
- removeAllRights: all rights for removing the DRM plug-in.

4.5.3.2 Decryption Related APIs in DrmManagerClient.h

- openDecryptSession: used to open a decryption session for decrypting protected contents.
- closeDecryptSession: used to close the decryption session corresponding to the handle.

- initializeDecryptUnit: used to decrypt a decryption unit and set the DRM header data.
- finalizeDecryptUnit: used to finalize initialization of a decryption unit.
- consumeRights: right for consuming the content.
- Pread: used to read specified byte data from the DRM file. The PlayReady plug-in does not support this interface.
- decrypt: used for decryption.

4.5.4 Right Acquisition Related APIs in DrmManagerClient.java

getMetadata

[Description]

Obtain the metadata of the content.

[Syntax]

```
ContentValues getMetadata(URI uri);
```

[Parameter]

Parameter	Description	Input/Output
uri	Uniform resource identifier (URI) of the content.	Input

[Return Value]

Return Value	Description
ContentValues Sample	Indicates the key-value pair of the metadata. The PlayReady plug-in supports the following metadata key: Key = "CONTENT_MIME_TYPE", value = MIME type of the content with the specified URI.
NULL	Failed.

[Requirement]

Class: android.drm.DrmManagerClient;

getOriginalMimeType

[Description]

Obtain the MIME type of the original content.

[Syntax]

```
String getOriginalMimeType(String path);
```

[Parameter]

Parameter	Description	Input/Output
path	Path of the protected contents.	Input

[Return Value]

Return Value	Description
MIME type	MIME type of the protected contents, such as video/mpeg.
NULL	Failed.

[Requirement]

Header: android.drm.DrmManagerClient;

canHandle

[Description]

Check whether the MIME type or content is supported.

[Syntax]

```
boolean canHandle(String path, String mimeType);
```

[Parameter]

Parameter	Description	Input/Output
path	Path of the protected contents.	Input
mimetype	MIME type of the object.	Input

[Return Value]

Return Value	Description
true	The plug-in is able to process the specified MIME type or path.
false	The plug-in cannot process the specified MIME type or path.

[Requirement]

Header: android.drm.DrmManagerClient;

acquireRights

[Description]

Execute the full right acquisition process, including communication with the license server.

[Syntax]

```
int acquireRights(DrmInfoRequest drmInfoRequest);
```

[Parameter]

Parameter	Description	Input/Output
drmInfoRequest	Right acquisition request information.	Input

Table 4-3 DrmInfoRequest Custom Data – License Acquisition for AcquireDrmInfo

Info Type	Key	Value	Description
TYPE_RIGHTS_ACQUISITION_INFO	Action	AcquireLicenseFull	Automatic right acquisition request, communicating with the license server using the plug-in. Required parameter **key-value**: "Header" = <drm header> "CustomData" = custom data for license

[Return Value]

Return Value	Description
ERROR_NONE	Succeeded.
ERROR_UNKNOWN	Failed.

[Requirement]

Header file: android.drm.DrmManagerClient

acquireDrmInfo

[Description]

Return registration, deregistration, and right request information.

[Syntax]

```
DrmInfo acquireDrmInfo(DrmInfoRequest drmInfoRequest);
```

[Parameter]

Parameter	Description	Input/Output
drmInfoRequest	Right acquisition request information.	Input

Table 4-4 DrmInfoRequest Custom Data – License Acquisition for AcquireDrmInfo

Info Type	Key	Value	Description
TYPE_RIGHTS_ACQUISITION_INFO	Action	GenerateLicChallenge	Request for generating challenge data for right acquisition. Required parameter **key-value**: "Header" = <drm header> or "Data" = <filepath> "CustomData" = custom data for challenge Returned information **key-value**: "LA_URL" = url of license server "Data" = license challenge buffer
TYPE_RIGHTS_ACQUISITION_INFO	Action	ProcessLicResponse	Request for processing response data for server right acquisition. Required parameter **key-value**: "Data" = response buffer Returned information **key-value**: "Data" = License Ack challenge buffer
TYPE_RIGHTS_ACQUISITION_INFO	Action	ProcessLicAckResponse	Request for processing server ACK response data. Required parameter **key-value**: "Data" = ack response buffer
TYPE_RIGHTS_ACQUISITION_INFO	Action	GenerateMeterCertChallenge	Request for generating challenge data of the metering certificate. Required parameter **key-value**: "MeteringId" = metering ID "CustomData" = custom data for challenge Returned information **key-value**: "Data" = challenge buffer
TYPE_RIGHTS_ACQUISITION_INFO	Action	ProcessMeterCertResponse	Request for processing response data of the server metering certificate. Required parameter **key-value**: "MeteringId" = metering ID "CustomData" = custom data for challenge Returned information **key-value**: "Data" = response buffer
TYPE_RIGHTS_ACQUISITION_INFO	Action	GetDrmHeader	Request for obtaining WRMHEADER data of the local content. Required parameter **key-value**: "path" = path to content Returned information **key-value**: "Data" = drm header buffer

Info Type	Key	Value	Description
TYPE_RIGHTS_ACQUISITION_INFO	Action	GenerateMeterDataChallenge	Request for generating challenge data of the metering data. Required parameter **key-value**: "MeteringId" = metering ID "CustomData" = custom data for challenge Returned information **key-value**: "Data" = challenge buffer
TYPE_RIGHTS_ACQUISITION_INFO	Action	ProcessMeterDataResponse	Request for processing server metering data response data. Required parameter **key-value**: "Data" = response buffer
TYPE_REGISTRATION_INFO	Action	ProcessJoinDomInitiator	Request for processing the **jion-domain initiator** file. Initiator file example: http://playready.directtaps.net/pr/initiator.aspx?p=0&type=JOIN Required parameter **key-value**: "Data" = initiator text (String)
TYPE_REGISTRATION_INFO	Action	GenerateJoinDomChallenge	Request for generating challenge data of the **Join Domain**. Usually, the challenge data is sent to the domain controller URL address, which is recorded in the initiator file. Required parameter **key-value**: "DS_ID" "AccountID" "Revision" "FriendlyName" "CustomData" Returned information **key-value**: "Data" = challenge buffer
TYPE_REGISTRATION_INFO	Action	ProcessJoinDomResponse	Request for processing Join Domain response data. Required parameter **key-value**: "Data" = response buffer
TYPE_REGISTRATION_INFO	Action	ProcessLeaveDomInitiator	Request for processing the **leave domain initiator** file. Initiator file example: http://playready.directtaps.net/pr/initiator.aspx?p=0&type=LEAVE Required parameter **key-value**:

Info Type	Key	Value	Description
			"Data" = initiator text (String)
TYPE_REGISTRATION_INFO	Action	Get_OPL_Miracast_Values	Returns the OPL and Miracast value. Only after contents are bound to a license, the OPL and Miracast value can be set. Therefore, this action is valid only when **ConsumeRights()** is called during decryption. Currently, the OPL and Miracast enabler value can be returned for only one decryption session. The OPL value returned upon **AcquireDrmInfo()** is a number such as **300**. When the **PlayReady License** contains the **Miracast Play Enabler Type** object and the domain value of **Play Enabler Type** is **{A340C256-0941-4D4C-AD1D-0B6735C0CB24}**, the value of **DRM_MIRACAST_ENABLER** is **true**; otherwise, the value is **false**. Note: 1. The returned OPL and Miracast values are related to only the current decryption session. 2. The **openDecryptSession()** and **AcquireDrmInfo()** must use the same instance **DrmManagerClient**. Returned value **key-value**: OPL_AUDIO_COMPRESSED_LEVEL OPL_AUDIO_UNCOMPRESSED_LEVEL OPL_VIDEO_COMPRESSED_LEVEL OPL_VIDEO_UNCOMPRESSED_LEVEL OPL_VIDEO_ANALOG_LEVEL DRM_MIRACAST_ENABLER
TYPE_UNREGISTRATION_INFO	Action	GenerateLeaveDomChallenge	Request for generating challenge data of the **Leave Domain**. Required parameter **key-value**: "DS_ID" "AccountID" "Revision" "CustomData" Returned information **key-value**: "Data" = challenge buffer

Info Type	Key	Value	Description
TYPE_UNREGISTRATION_INFO	Action	ProcessLeaveDomResponse	Request for processing Leave Domain response data. Required parameter **key-value**: "Data" = response buffer
TYPE_UNREGISTRATION_INFO	Action	RemoveAllLicenses	Request for removing all PlayReady licenses.
TYPE_UNREGISTRATION_INFO	Action	isRemoveAllLicenses	Checks whether **RemoveAllLicenses** is supported. If yes, **"Status" = "ok"** is returned for **acquireDrmInfo**.

[Return Value]

Return Value	Description
ERROR_NONE	Succeeded.
ERROR_UNKNOWN	Failed.

[Requirement]

Header file: android.drm.DrmManagerClient;

getConstraints

[Description]

Return right constraint information of the content.

[Syntax]

```
ContentValues getConstraints(URI uri, int action);
```

[Parameter]

Parameter	Description	Input/Output
uri	URI of the content.	Input
action	Actions can be defined as "Action::DEFAULT" or "Action::PLAY". This parameter will not be used.	Input

[Return Value]

Return Value	Description
ContentValues Sample	Indicates the key-value pair of the constraint value. The PlayReady plug-in supports the following constraints: • DrmConstraints: MAX_REPEAT_COUNT Maximum number of repetition times. • DrmConstraints: REMAINING_REPEAT_COUNT Remaining number of repetition times. • DrmConstraints: LICENSE_START_TIME Right start time, before which protected contents cannot be played. The data unit is second. The time starts from January 1, 1970. • DrmConstraints: LICENSE_EXPIRY_TIME Right end time, after which protected contents cannot be played. The data unit is second. The time starts from January 1, 1970. • DrmConstraints: LICENSE_AVAILABLE_TIME Right valid time, applicable to the FIRST_PLAY_EXPIRATION right. • DrmConstraints: EXTENDED_METADATA Specifies whether the content can be played as a ring tone.
NULL	Failed.

[Requirement]

Header file: android.drm.DrmManagerClient;

checkRightsStatus

[Description]

For a specified action, check whether the right for the protected contents is valid.

[Syntax]

```
int checkRightsStatus(String path, int action);
```

[Parameter]

Parameter	Description	Input/Output
path	Path of the protected contents.	Input
action	Action to be executed. For example, Action::DEFAULT, Action::PLAY.	Input

The following table describes the relationship between actions and PlayReady operations.

DRM Framework Action	PlayReady Operation	Allowed or Not
DEFAULT	Play	Depending on the certificate
PLAY	Play	Depending on the certificate
RINGTONE	Ring tone	Depending on the certificate
OUTPUT	Undefined	Not allowed
PREVIEW	Undefined	Not allowed
EXECUTE	Execute an operation.	Depending on the certificate
DISPLAY	Undefined	Not allowed
TRANSFER	Copy contents to other devices.	Allowed

[Return Value]

Return Value	Description
RightsStatus::RIGHTS_VALID	The right is valid.
RightsStatus::RIGHTS_EXPIRED	The right expires.
RightsStatus::RIGHTS_INVALID	No right is available or an error occurs.

[Requirement]

Header file: android.drm.DrmManagerClient;

removeRights

[Description]

Right for removing the content.

[Syntax]

```
int removeRights(String path);
```

[Parameter]

Parameter	Description	Input/Output
path	Path of the protected contents. • The following character strings can be used to delete expired or other certificates: • All certificates: "playready://AllLicenses.playready" • All expired certificates: "playready://AllExpiredLicenses.playready" • Certificate for specified contents: "playready://xxx.playready". The xxx indicates the path of the contents.	Input

[Return Value]

Return Value	Description
ERROR_NONE	Succeeded.
ERROR_UNKNOWN	Failed.

[Requirement]

Header file: android.drm.DrmManagerClient;

removeAllRights

[Description]

All rights for removing the DRM plug-in, used to restore factory settings.

[Syntax]

```
int removeAllRights();
```

[Parameter]

Parameter	Description	Input/Output
None	None	

[Return Value]

Return Value	Description
ERROR_NONE	Succeeded.
ERROR_UNKNOWN	Failed.

[Requirement]

Header file: android.drm.DrmManagerClient;

4.5.5 Decryption Related APIs in DrmManagerClient.h

openDecryptSession

[Description]

Open a decryption session for decrypting protected contents.

[Syntax]

```
sp<DecryptHandle> openDecryptSession(int fd, off64_t offset, off64_t length,
const char* mime);
```

[Parameter]

Parameter	Description	Input/Output
fd	File descriptor of the protected content.	Input
offset	Start point of the content. This parameter will not be used.	Input
length	Length of the protected content. This parameter will not be used.	Input
mime	If the value is not NULL, set the MIME type of the content.	Input

[Return Value]

Return Value	Description
DecryptHandle	Handle of the decryption session. A value can be assigned to DecryptHandle by this interface. • decryptApiType = DecryptApiType::ELEMENTARY_STREAM_BASED • status = DRM_NO_ERROR • extendedData = Plug-in name (Key="pluginName") • mimeType If the input parameter **mime** of the API is not null, set the MIME type of the content to **mime**. If the parameter is null, set the MIME type of the content to the MIME type of the content. If the MIME type of the content cannot be obtained by the plug-in, set it to the default value **video/ismv**.
NULL	The decryption session cannot be opened.

[Requirement]

Header file: DrmManagerClient.h

[Note]

- This API is used to play the opened local file.
- This API cannot be used to play a smoothstreaming file (such as ismv/isma).

openDecryptSession

[Description]

Open a decryption session for decrypting protected local or online stream contents.

[Syntax]

```
sp<DecryptHandle> openDecryptSession(const char* uri, const char* mime);
```

[Parameter]

Parameter	Description	Input/Output
uri	Path of the protected contents. For online stream files, use the following fixed character strings: • "ms-smooth-streaming" • "http-stream-download"	Input
mime	If the value is not NULL, set the MIME type of the content.	Input

[Return Value]

Return Value	Description
DecryptHandle	Handle of the decryption session. • A value can be assigned to DecryptHandle by this interface. • decryptApiType = DecryptApiType::ELEMENTARY_STREAM_BASED • status = DRM_NO_ERROR • extendedData = Plug-in name (Key="pluginName") • mimeType If the input parameter mime of the API is not null, set the MIME type of the content to mime. If the parameter is null, for an online stream file, set the parameter to the default value **video/ismv**; for a local file, set the MIME type of the content to the MIME type of the content; if the MIME type of the content cannot be obtained by the plug-in, set the parameter to the default value **video/ismv**.
NULL	The decryption session cannot be opened.

[Requirement]

Header file: DrmManagerClient.h;

closeDecryptSession

[Description]

Close the decryption session corresponding to the handle.

[Syntax]

```
status_t closeDecryptSession(sp<DecryptHandle> &decryptHandle);
```

[Parameter]

Parameter	Description	Input/Output
decryptHandle	Handle of the decryption session.	Input

[Return Value]

Return Value	Description
DRM_NO_ERROR	Succeeded.
DRM_ERROR_UNKNOWN	Failed.

[Requirement]

Header file: DrmManagerClient.h;

initializeDecryptUnit

[Description]

Decrypt a decryption unit and set the DRM Header data.

[Syntax]

```
status_t initializeDecryptUnit(sp<DecryptHandle> &decryptHandle, int
decryptUnitId, const DrmBuffer* headerInfo);
```

[Parameter]

Parameter	Description	Input/Output
decryptHandel	Handle of the decryption session.	Input
decryptUnitId	Used to define the ID of a decryption unit, such as track ID. This parameter will not be used.	Input

Parameter	Description	Input/Output
headerInfo	WMDRM header information of the PlayReady file. Note that contents of the whole PlayReady Header Objects are contained. To obtain the standards of PlayReady Header Objects, download this file at: http://www.microsoft.com/playready/documents/	Input

[Return Value]

Return Value	Description
DRM_NO_ERROR	Succeeded.
DRM_ERROR_UNKNOWN	Failed.

[Requirement]

Header file: DrmManagerClient.h;

[Note]

This interface is called only when an online smoothstreaming file is played. When a local file is played, this interface cannot be used.

finalizeDecryptUnit

[Description]

Finalize initialization of a decryption unit.

[Syntax]

```
status_t finalizeDecryptUnit(sp<DecryptHandle> &decryptHandle, int
decryptUnitId);
```

[Parameter]

Parameter	Description	Input/Output
decryptHandel	Handle of the decryption session.	Input
decryptUnitId	Used to define the ID of a decryption unit, such as track ID. This parameter will not be used.	Input

[Return Value]

Return Value	Description
DRM_NO_ERROR	Succeeded.
DRM_ERROR_UNKNOWN	Failed.

[Requirement]

Header file: DrmManagerClient.h;

[Note]

This interface is called only when an online smoothstreaming file is played. When a local file is played, this interface cannot be used.

consumeRights

[Description]

Right for consuming the content.

[Syntax]

```
status_t consumeRights(sp<DecryptHandle> &decryptHandle, int action, bool reserve);
```

[Parameter]

Parameter	Description	Input/Output
decryptHandel	Handle of the decryption session.	Input
action	Action to be executed. For example, Action::DEFAULT, Action::PLAY.	Input
reserve	If this right should be reserved, set this parameter to **true**.	Input

[Return Value]

Return Value	Description
DRM_NO_ERROR	The right for consuming the content is obtained.
DRM_ERROR_UNKNOWN	The right for consuming the content cannot be obtained.
DRM_ERROR_NO_LICENSE	No right is available.
DRM_ERROR_LICENSE_EXPIRED	The right expires.
DRM_ERROR_CANNOT_HANDLE	The error cannot be handled

[Requirement]

Header file: DrmManagerClient.h;

[Note]

For each encryption session, this API can be called only once.

decrypt

[Description]

Decrypt protected contents.

[Syntax]

```
status_t decrypt(sp<DecryptHandle> &decryptHandle, int decryptUnitId, const
DrmBuffer* encBuffer, DrmBuffer** decBuffer, DrmBuffer* IV = NULL);
```

[Parameter]

Parameter	Description	Input/Output
decryptHandel	Handle of the decryption session.	Input
decryptUnitId	Decryption unit ID such as track ID. For the PlayReady HW plug-in, this parameter indicates the type of the data to be decrypted, audio or video. Video: The value of **Track ID** is **0**, and data is decrypted to the secure memory. Audio: The value of **Track ID** is **1**, and data is decrypted to a non-secure memory.	Input
encBuffer	Used to encrypt data buffer. For the PlayReady HW plug-in: • The hardware decryption module directly uses the physical address for decryption to avoid repeated data copy between the non-secure side and secure side. • **encBuffer** is responsible for sending the physical addresses of encrypted data buffer and decrypted data buffer to the PlayReady plug-in for processing. • The encrypted data buffer refers to buffer with no cache assigned by **HI_MMZ_Malloc()** to the non-secure side. • For video data, the decrypted data buffer refers to buffer with no cache assigned by **HI_SEC_MMZ_New()** to the secure side. For audio data, the decrypted data buffer refers to buffer assigned by **HI_MMZ_Malloc()** to the non-secure side. • The corresponding data structure of **encBuffer.data** is private data **PhysicalBuffer** as defined by **DrmExtractor.h**.	Input

Parameter	Description	Input/Output
decBuffer	Used to decrypt data buffer. For the PlayReady HW plug-in: For video data, the decrypted data buffer refers to buffer with no cache assigned by **HI_SEC_MMZ_New()** to the secure side. For audio data, the decrypted data buffer refers to buffer assigned by **HI_MMZ_Malloc()** to the non-secure side.	Output
IV	Initialization vector (IV) buffer. IV.data points to the private data structure DxMultiSampleHeader, which is followed by several DxSubSamples. The number of DxSubSamples is determined by **dwSubSamplesNum** in **DxMultiSampleHeader**.	Input

[Return Value]

Return Value	Description
DRM_NO_ERROR	Succeeded.
DRM_ERROR_UNKNOWN	Failed.
DRM_ERROR_SESSION_NOT_OPENED	The session is not opened.
DRM_ERROR_LICENSE_EXPIRED	The right expires.
DRM_ERROR_DECRYPT_UNIT_NOT_INITIALIZED	The decryption unit is not initialized.
DRM_ERROR_DECRYPT	A decryption error occurs.

[Requirement]

Header file: DrmManagerClient.h, DRMExtractor.h

[Note]

For details about the **decrypt()** function and its parameters used for the PlayReady HW plug-in, see the *Discretix Android DRM Framework PlayReady Plug-in for Recommended Plug-in User Manual*.

4.5.6 Summary of Data Types

This document provides only private data types that are used by decryption interfaces.

- PhysicalBuffer: A data structure defined by the PlayReady HW plug-in to transmit physical addresses of the encrypted and decrypted data buffers.
- DxMultiSampleHeader: Definition of IV and data structure header of multiple network abstraction layer (NAL) units.
- DxSubSample: Definition of the NAL data structure.

4.5.7 Description of Data Types

PhysicalBuffer

[Description]

A data structure defined by the PlayReady HW plug-in to transmit physical addresses of the encrypted and decrypted data buffers.

This data type is defined in **DrmExtractor.h**.

[Syntax]

```
typedef struct
{
    unsigned int encPhyAddr;
    unsigned int encVirAddr;
    int encDataSize;
    unsigned int decPhyAddr;
    unsigned int decVirAddr;
    int decDataSize;
} PhysicalBuffer;
```

[Member]

Member Name	Description
encPhyAddr	Physical address of the encrypted data buffer.
encVirAddr	Virtual address of the encrypted data buffer.
encDataSize	Length of the encrypted data.
decPhyAddr	Physical address of the decrypted data buffer.
decVirAddr	Virtual address of the decrypted data buffer. If the decrypted data buffer is on the secure side, this parameter is not used.
decDataSize	Length of the decrypted data.

DxMultiSampleHeader

[Description]

Define the IV and data structure header of multiple NALs.

[Syntax]

```
typedef struct
{
    uint64_t qwInitializationVector;
    uint32_t dwOutBufferOffset;
```

```
    uint32_t dwMediaOffset;
    uint32_t dwSubSamplesNum;
} DxMultiSampleHeader;
```

[Member]

Member Name	Description
qwInitializationVector	8-byte IV obtained from a **piff** or an **asf** file.
dwOutBufferOffset	Offset address of the output buffer, serving as the start address for data copying or decrypting.
dwMediaOffset	Offset address of encrypted data, usually set to **0**. When the decryption function is called multiple times for decrypting multiple NALs in a data frame, the offset address is not **0**.
dwSubSamplesNum	Number of **DxSubSample** structures following **DxMultiSampleHeader**.

DxSubSample

[Description]

Define the NAL data structure.

[Syntax]

```
typedef struct
{
    uint32_t dwClearDataSize;
    uint32_t dwEncryptedDataSize;
} DxSubSample;
```

[Member]

Member Name	Description
dwClearDataSize	Size of the non-encrypted data to be copied from the encrypted data buffer to the decrypted data buffer.
dwEncryptedDataSize	Size of the data to be decrypted.

[Note]

Pay attention to the relationship between DxMultiSampleHeader and DxSubSample and NAL units.

Figure 4-5 shows the data structure of multiple NAL units.

Figure 4-5 Data structure of multiple NAL units