

# FairPlay Streaming Programming Guide

---

# Contents

<b>About FairPlay Streaming .....</b>	<b>5</b>
At a Glance .....	5
A Key Server Module Wraps a Key for Delivery .....	7
An FPS-Aware Playback App Asks the Key Server Module for a Key .....	7
FPS Sends Content Keys Securely .....	7
Content Server Delivers the Content Stream .....	9
FairPlay Streaming SDK Contents .....	9
See Also .....	9
<b>Programming the Key Server Module .....</b>	<b>10</b>
Overview of Processing Steps .....	10
Cryptographic Formula Syntax .....	10
SPC and CKC Messages .....	11
TLLV Block Structure .....	11
The SPC Message .....	13
The SPC Payload .....	13
Session Key and Random Value Block .....	16
Session Key and Random Value Integrity Block .....	17
Protocol Version Blocks .....	18
Constructing the CKC Message .....	20
CKC Payload .....	20
Encrypting the Content Key .....	21
Returning SPC Blocks in the CKC Payload .....	22
Encrypting the CKC Payload .....	22
<b>Understanding the TLLVs .....</b>	<b>24</b>
Capabilities .....	24
Device Identity .....	26
Device Info Support .....	28
Next Deny List .....	29
Security Level Support .....	30
HDCP Enforcement .....	33
Virtual Machine Support .....	34

<b>Renting and Leasing the Content Key .....</b>	<b>35</b>
Content Key Expiration .....	35
Video Rental .....	35
Secure Lease .....	35
TLLV for Rental and Lease .....	35
Establishing the Rental and Lease Period .....	36
Simultaneous Renting and Leasing .....	37
<b>Offline FairPlay Streaming .....</b>	<b>39</b>
Preloading Offline Keys .....	40
<b>Offline Rental Support .....</b>	<b>43</b>
Storage and Playback Expiry .....	43
Offline Key .....	43
Sync .....	44
<b>Developing an FPS-Aware App .....</b>	<b>47</b>
Identifying Your FPS App with an Application Certificate .....	47
Integrating FPS with the iOS Decryption Process .....	47
Integrating FPS in Safari .....	48
EME Message Exchange .....	48
Requesting a Content Key from the Key Server .....	49
Processing the Key Server's Response .....	49
Configuring AirPlay Mode .....	50
Interpreting Error Messages .....	50
Manually Fetching FPS Error Messages .....	51
<b>Formatting and Encrypting Streams .....</b>	<b>52</b>
Preparing Content for FPS .....	52
Including Initialization Vectors (IV) in Playlists .....	52
Using FPS Options with the Media File Segmenter .....	53
Using ALLOWED-CPC in playlist to improve tier selection .....	54
<b>Document Revision History .....</b>	<b>56</b>

# Figures

Figure 1-1 FPS exchanges .....	7
Figure 1-2 FPS information flow .....	8
Figure 2-1 TLLV block structure .....	11
Figure 5-1 Offline playback .....	40
Figure 5-2 Persistent key request .....	41
Figure 8-1 FPS information flow .....	49
Listing 8-1 Manually fetching FPS errors .....	51
Listing 9-1 Adding FPS to an HLS Playlist .....	52
Listing 9-2 Media File Segmenter command .....	53
Listing 9-4 Adding ALLOWED-CPC to an HLS Playlist .....	54

# About FairPlay Streaming

Apple FairPlay Streaming (FPS) securely delivers keys to:

- Apple mobile devices (iOS-based, iPadOS-based, watchOS-based, and visionOS-based).
- Apple TV.
- Safari on macOS, iPadOS and iOS.

This will enable playback of encrypted video content. This content is delivered over the Web using HTTP Live Streaming (HLS) technology.

For simplicity, these devices will collectively be referred to as “Apple devices” throughout the remainder of this document.

Note that FPS also delivers keys to FPS-enabled TVs and STBs via AirPlay protocol.

FPS protects the delivery of keys that decrypt streamed audio and video media. An Apple device can securely acquire a key from a content provider's key server. The operating system uses the key to decrypt the media before playback.

FPS key delivery offers the following features and behaviors:

- AES 128-bit content keys generated by the key server.
- Every key is known only to the key server and to the Apple device.
- When playback is stopped, the key for the Apple device is permanently discarded from memory.
- The key server can specify the duration of the key's validity for the Apple device.
- Protection of MPEG-2 file formats.

FPS allows the device to stop playback based on expiration information sent with the content key. Using FPS on an Apple device ensures secure key transmission and secure use for media decryption.

## At a Glance

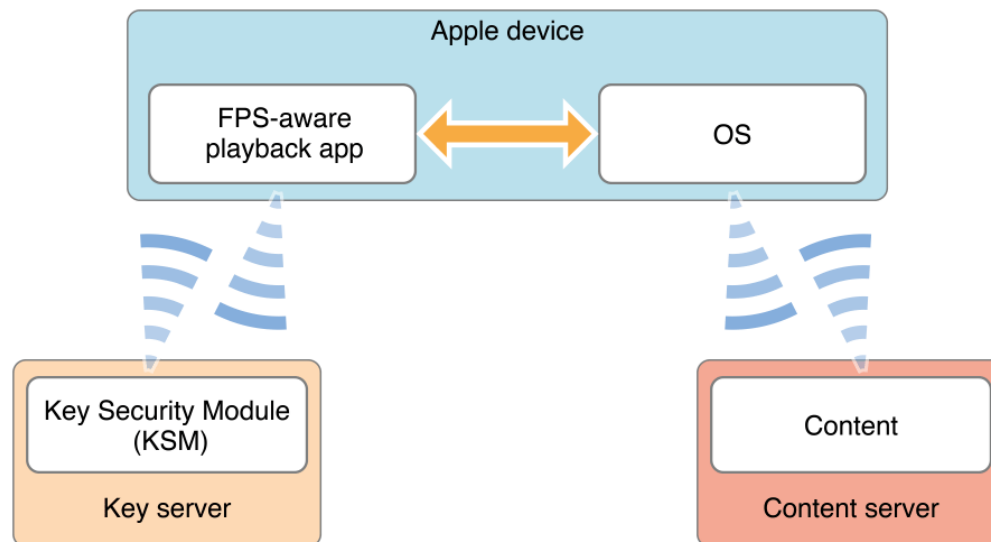
As an approved Apple FPS developer, you implement FPS by writing code to run on your key server and in your playback app, so that both recognize FPS messages. When an Apple device plays a media stream with a playlist containing an FPS-specific tag, the operating system asks the app to obtain the decryption key. To do so, the app calls an API that invokes FPS, causing the operating system to prepare an encrypted request for the key for that media. When the app sends the request to the server, the FPS code on the server wraps the required key in an encrypted message and sends it to the app. The app then asks the operating system to unwrap the message and decrypt the stream, so the Apple device can play the media.

The implementation process requires three programming tasks:

- Writing a *Key Server Module* that is installed in a key server's software. This module exchanges messages with the Apple device during the FPS process.

- Adding code to make an Apple device playback app *FPS-aware*. The app communicates with a server that can deliver the key to decrypt the content, such as a movie. Figure 1-1 shows an FPS system including an FPS-aware playback app.
- Creating the formatting and encryption software for the media content server. This software prepares the encrypted content stream according to the Apple HTTP Live Streaming (HLS) specification.

**Figure 1-1** FPS exchanges



### A Key Server Module Wraps a Key for Delivery

The Key Server Module (KSM) implements FPS algorithms that can interpret an encrypted key request message from an Apple device and create an encrypted response containing the key for the specified media. You must write code for the key server enabling those algorithms.

### An FPS-Aware Playback App Asks the Key Server Module for a Key

A FPS-aware playback app uses an operating system programming interface to obtain the encrypted request for the key, send it to the KSM, and receive the key for the media asset decryption. Your app will also need code supporting user interactions.

### FPS Sends Content Keys Securely

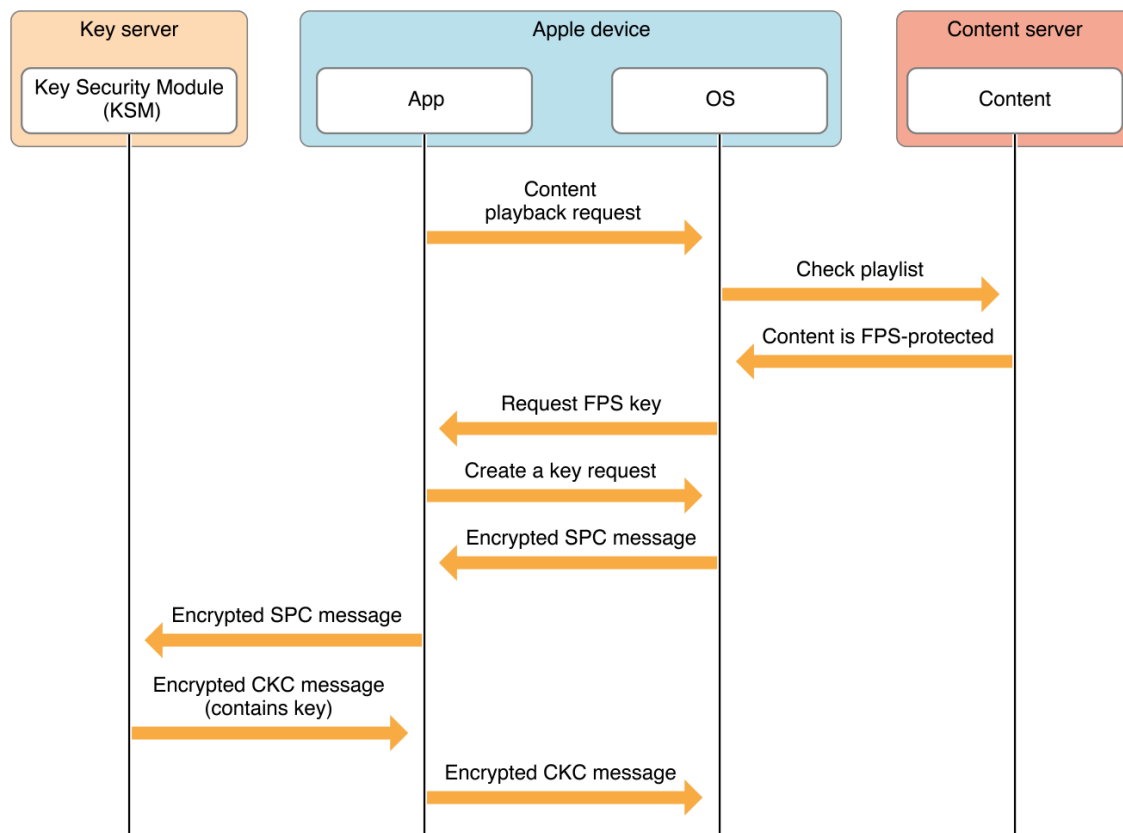
An FPS messaging session delivers a content key to the player app. Typically, the session proceeds as follows:

1. The app asks the operating system to play specific content identified by a URL.
2. The operating system accesses the content and checks its playlist.
3. An attribute in the playlist identifies the content as encrypted by a content key obtainable through FPS.
4. The operating system informs the app that the content is encrypted using FPS.
5. The app asks the operating system to prepare an FPS message that requests the content key.
6. The operating system delivers an encrypted Server Playback Context (SPC) message to the app.
7. The app sends the SPC to a key server that contains a KSM.
8. The KSM decrypts the SPC and gets the requested content key from the key server.

9. The KSM wraps the content key inside an encrypted content key context (CKC) message, which it sends to the app.
10. The app delivers the CKC to FPS software integrated in the operating system, which then uses it to decrypt the media content, as described below.

These steps move information between FPS modules as shown in Figure 1-2.

**Figure 1-2** FPS information flow



After the FPS software receives the CKC, it extracts the content key and provides the key to the OS. The OS uses the key to decrypt and play the content requested in step 1 of Figure 1-2.

The streaming media playlist contains a list of versions of FPS that the key server supports. The operating system discerns and lists the mutually recognized FPS versions in the operating system's encrypted key request. The key server then picks which version to use.

Throughout this process, the app and the server can communicate through any transport link chosen by the app developer.

The content streams in conformance with the HLS protocol, using H.264 video and audio formats.



## Content Server Delivers the Content Stream

The content server delivers the formatted and encrypted content stream to the Apple device. Only the Apple device, using keys delivered by FPS, can decrypt the stream and recover its content.

## FairPlay Streaming SDK Contents

The FairPlay Streaming Server SDK contains a reference implementation of the Key Server Module, a client sample, a specification and a set of test vectors. The test vectors can help establish and test the Key Server Module.

An FPS transaction round-trip between the server Key Server Module and a client requires production credentials. The content owner must request the FPS Deployment Package materials to complete the Key Server Module for production deployment.

## See Also

The following documents contain specifications and instructions that supplement the material in this programming guide:

- See [HTTP Live Streaming Overview](#) for general guidance on Apple streaming technology used with FairPlay Streaming.
- See [MPEG-2 Stream Encryption Format for HTTP Live Streaming](#) for information on the FairPlay Streaming media formats.
- See [HTTP Live Streaming Protocol](#) for the IETF Internet-Draft of the HLS specification.
- See the following industry standards, relevant to FPS:
  - [Information technology—generic coding of moving pictures and associated audio information: Systems](#) is the ITU-T Recommendation H.222.0 document, also published as [ISO/IEC International Standard 13818-1:2013](#).
  - [Advanced video coding for generic audiovisual services](#) is the ITU-T Recommendation H.264 document, also published as [ISO/IEC International Standard 14496-10:2014](#).
  - [Information technology—Coding of audio-visual objects—Part 3: Audio](#) is the [ISO/IEC International Standard 14496-3:2009](#).
  - [Digital Audio Compression Standard \(AC-3\)](#) is the Advanced Television Systems Committee (ATSC) standard A/52:2012.

# Programming the Key Server Module

The Key Server Module (KSM) is the part of FairPlay Streaming (FPS) technology that resides in the software of a content provider's key server. Its code must run on the server platform and implement the algorithms described in this chapter.

The KSM serves as a liaison between the playback app and the Apple device. The initial message from app to Apple device contains the server playback context (SPC). The Apple device's operating system parses the SPC and generates the content key context (CKC). The KSM encrypts and delivers the content key to the Apple device. The Apple device uses the content key to decrypt the FPS media sent from the content server. [Requesting a Content Key from the Key Server](#) describes this SPC exchange.

## Overview of Processing Steps

Table 2-1 summarizes a typical sequence of actions that a server and its KSM might perform to support FPS.

**Table 2-1 Typical server program steps**

Step	Server action
1	Receive an SPC message from an app running on an Apple device and parse it. See <a href="#">The SPC Message</a> .
2	Check the SPC's certificate hash value against the AC. See <a href="#">Identifying Your FPS App with an Application Certificate</a> and Table 2-3.
3	Decrypt the SPC payload. See <a href="#">SPC Payload Decryption</a> .
4	Verify that the Apple device is using a supported version of FPS software. See <a href="#">Protocol Version Blocks</a> .
5	Decrypt the session key and random value block in the SPC payload. See <a href="#">Decrypting the [SK...R1] Payload</a> .
6	Check the integrity of the SPC message. See <a href="#">Session Key and Random Value Integrity Block</a> .
7	Encrypt the content key. See <a href="#">Encrypting the Content Key</a> .
8	Assemble the contents of the CKC payload. See Table 2-11.
9	Encrypt the CKC payload. See <a href="#">Encrypting the CKC Payload</a> .
10	Construct the CKC message and send it to the app on the Apple device. See Table 2-10.

## Cryptographic Formula Syntax

This chapter uses the following conventions to formulate cryptographic processes:

- Square brackets denote an encrypted value. For example, [Info] means that the plaintext value Info is encrypted.

- $e(\text{Info})_K$  denotes the encryption of the plain text value `Info`, using the key  $K$ . The convention may also specify the encryption algorithm:  $\text{RSA } e(\text{Info})_K$  indicates the use of RSA encryption, whereas  $\text{AES\_CBC}_{IV} e(\text{Info})_K$  indicates the use of AES encryption in cipher block chaining (CBC) mode with an initialization vector (IV).
- Similarly,  $d(\text{Info})_K$  denotes the decryption of the encrypted value `Info` using the key  $K$ , and  $\text{AES\_ECB } d(\text{Info})_K$  indicates the use of AES decryption in electronic codebook (ECB) mode.

For an example of this syntax, see [SPC Payload Decryption](#).

## SPC and CKC Messages

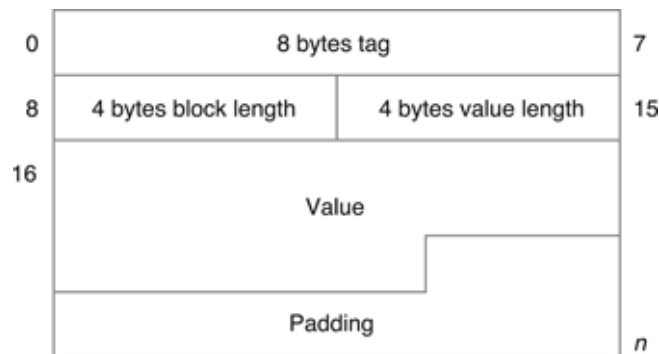
The SPC message that the playback app sends to the key server, and the CKC message that the KSM generates in reply, have these common characteristics:

- Messages consist of a fixed-length header followed by a variable-length payload.
- Payload is encrypted; in the SPC, part of the header is encrypted as well.
- The payloads in both the SPC and the CKC are divided into structures called tag-length-length-value (TLLV) blocks. This data layout is described in [TLLV Block Structure](#).
- TLLV blocks are tightly packed into the payload fields, but the blocks are located in random sequence.
- TLLV blocks are locatable in each payload by searching for their unique 8-byte tags, which begin each block. For each tag value, only one block with that tag can exist in an SPC or CKC message.
- The contents of TLLV blocks and all other FPS data structures are in cleartext AES format.
- All numeric fields in the SPC, CKC, and TLLVs are stored in network (big-endian) order.

### TLLV Block Structure

All TLLV blocks use the basic structure shown in Figure 2-1. The fields in this structure are in Table 2-2.

**Figure 2-1** TLLV block structure



**Table 2-2 TLLV block fields**

Field content	Byte range	Description
Tag	0-7	A sequence of bytes that is unique within an SPC or CKC payload.
Block length	8-11	The number of bytes in the value plus padding fields of the TLLV (following the tag, block length, and value length fields). The block length must be filled out to a multiple of 16 bytes by extending the padding field.
Value length	12-15	The number of bytes in the value field. This number may be any amount, including <code>0x0000</code> .
Value	16 ... <i>k</i>	The payload of the TLLV, starting with byte 16 of the block.
Padding	<i>k</i> +1 ... <i>n</i> ( <i>padding_size</i> )	<p>A field that begins with the next byte after the value field (byte <i>k</i>+1). It must fill out the TLLV to a multiple of 16 bytes, but may be randomly extended in increments of 16 bytes. Thus the following relation holds:</p> $padding\_size = block\_length - value\_length$ <p>The padding field must contain random values, not all <code>0x00</code> or <code>0xFF</code> bytes.</p>

**Note:** An SPC message may contain reserved TLLV blocks with tag values not covered in this documentation. Such blocks should be ignored by the KSM.

## The SPC Message

Configure the key server so that it delivers to its KSM every SPC message that the Apple device generates. Each SPC message is a container with a header of fixed-length fields and a variable-length data payload, as listed in Table 2-3. A sample SPC message is included in the FPS development support package; see [Using the FPS SDK and Tools](#).

**Table 2-3 SPC container structure**

Field content	Size (bytes)	Description
SPC version	4	The version number of the SPC. There are two supported versions: 0x00000001 uses RSA-1024 0x00000002 uses RSA-2048 Other than the encryption type and key size, versions 1 and 2 are identical.
Reserved	4	Reserved for Apple; ignore these bytes.
SPC data initialization vector (IV)	16	A CBC initialization vector that has a unique value for each SPC message. See <a href="#">SPC Payload Decryption</a> .
Encrypted AES-128 key	128 (version 1) or 256 (version 2)	The key for decrypting the SPC payload. This key is itself encrypted, using RSA public key encryption with Optimal Asymmetric Encryption Padding (OAEP), as described in <a href="#">SPC Payload Decryption</a> .
Certificate hash	20	The SHA-1 hash value of the encrypted Application Certificate, which identifies the private key of the developer that generated the SPC. See <a href="#">Identifying Your FPS App with an Application Certificate</a> .
SPC payload length	4	The number of bytes in the encrypted SPC payload. Because the payload consists of blocks whose lengths are multiples of 16 bytes, this number is a multiple of 16.
SPC payload	SPC payload length	A variable-length set of TLLV blocks, as described in <a href="#">TLLV Block Structure</a> . The whole payload is AES-128 encrypted using the encrypted key contained in bytes 24-151 of the SPC message. The minimum set of TLLV blocks that the KSM must extract from this payload is specified in <a href="#">SPC Payload Contents</a> .

### The SPC Payload

The SPC payload must be decrypted as specified in [SPC Payload Decryption](#). Read about the decrypted value in [SPC Payload Contents](#).

## SPC Payload Decryption

Decrypt the payload of the SPC message using the AES-128 cryptography standard with a cipher block chaining (CBC) mode of operation. The first block of the chain contains the CBC initialization vector (IV) contained in bytes 8-23 of the SPC message.

Obtain the key for the AES decryption of the SPC payload (called SPCK) by decrypting the “Encrypted AES-128 key” using the RSA cryptography standard with Optimal Asymmetric Encryption Padding (OAEP). The key for the RSA decryption of the SPCK is the server's RSA private key. An example of such a key is in the FPS developer support package; see [Using the FPS SDK and Tools](#).

In cryptographic formula syntax, decrypting the payload of the SPC consists of the following two decryption processes:

$$SPCK = RSA\_OAEP\_d([SPCK])_{Prv} \text{ where}$$

[SPCK] represents the value of SPC message bytes 24-151 for version 1 or 24-279 for version 2.  
Prv represents the server's private key.

$$SPC \text{ payload} = AES\_CBC_{IV}([SPC \text{ data}])_{SPCK} \text{ where}$$

[SPC data] represents the remaining SPC message bytes.  
IV represents the value of SPC message bytes 8-23.

## SPC Payload Contents

The decrypted SPC payload contains two TLLV types:

- Defined TLLVs listed in Table 2-4
- Undefined TLLVs

Any TLLV must appear only once in the SPC payload. TLLVs cannot be repeated in the same SPC payload.

**Table 2-4 TLLV blocks in the SPC payload**

TLLV content	Tag value	Description
[SK...R1]	0x3d1a10b8bffa2ec	A combination of values that the KSM will use to encrypt the content key and the CKC payload. Parse and decrypt this block as described in <a href="#">Session Key and Random Value Block</a> . Return the R1 value to FPS in the payload of the CKC message, as described in <a href="#">CKC Payload</a> .
[SK...R1] integrity	0xb349d4809e910687	A 16-byte value used to check the integrity of the contents of the [SK...R1] block. See <a href="#">Session Key and Random Value Integrity Block</a> .
Anti-replay (AR) seed	0x89c90f12204106b2	A 16-byte value used in the encryption of the CKC payload. See <a href="#">Encrypting the CKC Payload</a> .
R2	0x71b5595ac1521133	A 21-byte value used in decrypting the payload of the [SK...R1] block. See <a href="#">Decrypting the [SK...R1] Payload</a> .

TLLV content	Tag value	Description
Tag return request	0x19f9d4e5ab7609cb	A TLLV block that contains zero or more concatenated 8-byte values, each of which is the tag for a different TLLV block in the SPC. Retrieve and return all of the TLLV as is in the CKC payload. See <a href="#">Returning SPC Blocks in the CKC Payload</a> .
Asset ID	0x1bf7f53f5d5d5a1f	A content provider ID that tells the key server which content needs to be decrypted. The playback app may generate this value, or the FPS implementer may create it. Its length can range from 2 to 200 bytes, inclusive. The asset ID content is padded to a multiple of 16 bytes, regardless of the original length.
Transaction ID	0x47aa7ad3440577de	An 8-byte value that identifies the current FPS transaction. The KSM does not need to process this information.
Protocol versions supported	0x67b8fb79ecce1a13	A concatenation of 4-byte values identifying the Apple device-supported versions of FPS. See <a href="#">Protocol Version Blocks</a> .
Protocol version used	0x5d81bcbcc7f61703	A 4-byte value that identifies the version of FPS that the Apple device is using for this FPS transaction. See <a href="#">Protocol Version Blocks</a> .
Streaming indicator	0xab0256a31843974	A single 8-byte value. See Table 2-5.
Media playback state	0xeb8efdf2b25ab3a0	Media playback information for rental and lease. See <a href="#">TLLV for Rental and Lease</a> .
Capabilities	0x9c02af3253c07fb2	Apple device capabilities. See <a href="#">Capabilities</a> .
Device Identity	0x94c17cd676c69b59	Client device information. See <a href="#">Device Identity</a> .
Device Information	0xd43fc6abc596aae7	Legacy Client device information. See <a href="#">Device Info Support</a> .
Sync	0x77966de1dc1083ad	Used to synchronize expirations for downloaded rentals. See <a href="#">Offline Rental Support</a> .
Kext Deny List Version	0x70eca6573388e329	Client device Kext Deny List version. See <a href="#">Kext Deny List</a> .
Security Level Report	0xb18ee16ea50f6c02	Client device security level. See <a href="#">Security Level Support</a> .
VM Device Information	0x756440e240499f70	Virtual Machine device information. See <a href="#">Virtual Machine Support</a> .

**Table 2-5 Streaming indicator values**

Value	Description
0xab0256a31843974	AirPlay will send content to an Apple TV box or FPS-enabled TV or STB.
0x5f9c8132b59f2fde	An Apple digital AV adapter will send content.
Any other value	The requesting device plays back the content.

### Session Key and Random Value Block

The SPC delivers these two values in a single TLLV block called the [SK...R1] block:

- The 16-byte session key, SK, which is used to encrypt the content key as described in [Encrypting the Content Key](#).
- A 44-byte random number, R1, which is used in the encryption of the CKC payload as described in [Encrypting the CKC Payload](#). These bytes are also returned to the Apple device in the CKC payload, as shown in Table 2-11.

The structure of the whole [SK...R1] block, including its tag and length fields, is listed in Table 2-6. Decrypt the payload contained in this block as described in [Decrypting the \[SK...R1\] Payload](#).

**Table 2-6 [SK...R1] TLLV block**

Field content	Byte range	Description
TLLV tag	0-7	An 8-byte value of 0x3d1a10b8bfbfac2ec.
Total length	8-11	The total length of this TLLV block in bytes. The amount of padding at the end of the block, if any, determines the total length. It must be a multiple of 16 and greater than 127.
Value length	12-15	The length of the content of this TLLV block in bytes, 0x00000070 (decimal 112). The content consists of the initialization vector and payload fields.
Initialization vector (IV)	16-31	A 16-byte CBC initialization vector used in decrypting the next 96 bytes; see <a href="#">Decrypting the [SK...R1] Payload</a> .
Payload	32-127	The 96-byte payload of the block. Decrypt this payload as described in <a href="#">Decrypting the [SK...R1] Payload</a> to yield its contents.
Padding	128- <i>n</i>	Random values to fill out the TLLV to a multiple of 16 bytes. See the description of the <code>Padding</code> field in Table 2-2.



## Decrypting the [SK...R1] Payload

To recover the SK and the R1 value, the KSM must decrypt the payload of the block listed in Table 2-6, using AES-128 decryption with cipher block chaining (AES\_CBC). The KSM must use the initialization vector (IV) contained in bytes 16-31 of the [SK...R1] block (see Table 2-6) to initialize the first block of the AES\_CBC chain.

**Note:** Starting with FairPlay Streaming SDK Version 5, the ASk and D Function computation has been replaced with provisioning data and a precompiled cryptographic library.

Obtain the DAsK by following the instructions in *D Function Computation Guide*, a separate document that provides instructions on calculating the DAsK. The computation of the DAsK requires input of the R2 block contents and the application secret key (ASk). The DAsK value differs for each SPC request.

In cryptographic formula syntax, decrypt the payload of the [SK...R1] block using the following process:

$DAsK = D(R2, ASk)$  where

R2 represents the contents of the R2 block in the SPC payload.

ASk represents the playback app's secret key.

D represents the function described in *D Function Computation Guide*.

$payload = AES\_CBC_{IV} d([SK...R1] payload)_{DAsK}$  where

IV represents the value of [SK...R1] block bytes 16-31.

[SK...R1] payload represents the value of [SK...R1] block bytes 32-127.

SK and R1 are integrity numbers that represent fields in payload.

**Table 2-7 Decrypted [SK...R1] payload**

Field content	Byte range	Description
Session key (SK)	0-15	A 16-byte value used in the encryption of the content key. See <a href="#">Encrypting the Content Key</a> .
HU	16-35	A 20-byte value that represents the anonymized unique ID of the playback device. However, if the value of the streaming indicator TLLV (Table 2-4) in the SPC payload is 0x5f9c8132b59f2fde, then this value represents the ID of the Apple digital AV adapter and should not be used for device management (see Table 2-5).
R1	36-79	A 44-byte random number used in the encryption of the CKC payload and returned to the Apple device in the CKC payload. See <a href="#">Encrypting the CKC Payload</a> .
Integrity bytes	80-95	16 bytes used to check the integrity of this SPC message, as explained in <a href="#">Session Key and Random Value Integrity Block</a> .

## Session Key and Random Value Integrity Block

The [SK...R1] integrity block contains a 16-byte value that is used to check the integrity of the SPC. The KSM should compare the SPC contents with the 16-byte integrity number value in bytes 80-95 of the

decrypted [SK...R1] payload; see Table 2-7. If the two numbers are not identical, the SPC is not valid and the KSM should reject it.

Protocol Version Blocks

The integrated FPS code in each Apple device uses two TLLV blocks in the SPC (listed in Table 2-4) to tell the KSM about the device versioning.

- The protocol versions supported block lists all the versions of FPS that the Apple device supports.
- The protocol version used block identifies the one version that the Apple device is using for the current transaction.

The purpose of sending versioning information in the SPC is to ensure that the key server and the Apple device are using the same version of FPS, and that it is the latest version that they both support.

The streaming content’s playlist contains a list of the FPS versions that the key server supports for that content. As a good practice, your KSM should compare this list with the list of FPS versions in the protocol versions supported block. The protocol version used block should contain the ID of the most recent version common to both platforms. If the Apple device is not using the most recent common version, the app may be trying to attack FPS security. If there is no common version, the Apple device should not have generated an SPC and the KSM should reject the transaction.

Table 2-8 displays some recommended version configurations.

**Note:** Versioning should be decided between the server and the Apple device. The playback app should never contain embedded version information.

Table 2-8 Version configurations that follow good practices

Server	Apple device	SPC information
Version 1	Version 1	Used = 1 Supported = 1
Versions 1 and 2	Version 1	Used = 1 Supported = 1
Versions 1 and 2	Versions 1 and 2	Used = 2 Supported = 1 and 2
Version 2	Versions 1 and 2	Used = 2 Supported = 1 and 2

Table 2-9 displays other configurations that do not follow good practices and why these configurations should be avoided.

**Table 2-9 Version configurations that do NOT follow good practices**

Server	Apple device	SPC information	Implications
Version 1	Versions 1 and 2	Used = 1 Supported = 1 and 2	The server verifies the version used and uses version 1. However, a newer version of FPS is available, so the server should be updated.
Version 2	Version 1	Used = 1 Supported = 1	A mismatch exists between the FPS version on the Apple device and on the server; reject the SPC.
Versions 1 and 2	Versions 1 and 2	Used = 1 Supported = 1 and 2	An app may have tried to exploit the server by forcing it to use an old version of FPS.

## Constructing the CKC Message

The KSM must respond to every SPC message by returning a corresponding CKC message to the Apple device that sent it. Each CKC message is a container with a header of fixed-length fields and a variable-length data payload, as listed in Table 2-10. The *FPS Developer Support Package* packaged with the SDK contains a sample CKC message.

**Table 2-10 CKC container structure**

Field content	Byte range	Description
CKC version	0-3	The version number of the CKC. The version number covered by this programming guide is 0x00000001.
Reserved	4-7	Reserved by Apple; ignore these bytes.
CKC data initialization vector	8-23	A random 16-byte initialization vector, generated by the KSM, that has a unique value for each CKC message. The vector assists in initializing the first block of the AES_CBC chain, as described in <a href="#">Encrypting the CKC Payload</a> .
CKC payload length	24-27	The number of bytes in the encrypted CKC payload. Because the payload consists of blocks whose lengths are multiples of 16 bytes, this number is a multiple of 16.
CKC payload	28 ... <i>n</i>	A variable-length set of contiguous TLLV blocks, as described in <a href="#">CKC Payload</a> . The CKC payload is AES-128 encrypted as described in <a href="#">Encrypting the CKC Payload</a> .

### CKC Payload

The KSM uses the session key (SK) to encrypt the content key. The payload of a CKC message contains the content key that the Apple device uses to decrypt the media for playback.

Table 2-11 lists the TLLV blocks in the CKC payload. The order of these blocks should be random.

**Table 2-11 Contents of the CKC payload**

TLLV content	Tag value	Description
Encrypted CK	0x58b38165af0e3d5a	Mandatory. A TLLV block containing a content initialization vector and a 16-byte encryption of the content key provided by the server. See <a href="#">Encrypting the Content Key</a> .
R1	0xea74c4645d5efee9	Mandatory. A TLLV block containing the 44-byte R1 value that the KSM receives in the SPC payload. See <a href="#">Session Key and Random Value Block</a> .

TLLV content	Tag value	Description
Content key duration	0x47acf6a418cd091a	A TLLV that specifies the period of validity of the content key. This TLLV may be present only if the KSM receives an SPC with a Media Playback State TLLV. See <a href="#">Establishing the Rental and Lease Period</a> .
Blocks specified by a tag return request		The CKC must return, unchanged, the TLLV blocks that the SPC requested in a tag return request. See <a href="#">Returning SPC Blocks in the CKC Payload</a> .
Required Security Level	0x644cb1dac0313250	An optional TLLV, that specifies the minimum required Security Level of the client device. <a href="#">See Security Level Support</a> .
Offline Key	0x6375d9727060218c	Support for offline playback. See <a href="#">Offline Rental support</a> .
HDCP enforcement	0x2e52f1530d8ddb4a	An optional TLLV that specifies whether HDCP enforcement is required. The absence of this TLLV enforces HDCP Type 0. See Table 3-12.

Because all the blocks listed above are padded to multiples of 16 bytes, the CKC payload as a whole does not require further padding.

## Encrypting the Content Key

The content provider creates the content key that is used to decrypt the media on the Apple device. The provider must encrypt this key using AES-128 encryption before placing it into the CKC payload. The session key that FPS sent to the KSM in the SPC payload serves as the encryption key.

In crypto-formula syntax, encrypting the content key consists of the following process:

$[CK] = \text{AES\_ECB } e(CK)_{SK}$  where  
 CK is the content key provided by the key server.  
 SK is the content of the session key block from the SPC payload.

The encrypted content key must be 16 bytes long. It becomes the content of the content key TLLV, shown in Table 2-12, which is encrypted and made part of the CKC payload (see Table 2-11). The CKC payload is further encrypted as described in [Encrypting the CKC Payload](#).

**Table 2-12 Content Key TLLV**

Field name	Byte range	Description
TLLV tag	0-7	An 8-byte value of 0x58b38165af0e3d5a.
Total length	8-11	The total length of this TLLV block in bytes. The total length is determined by the amount of padding at the end of the block, if any; it must be a multiple of 16 and greater than 31.

Field name	Byte range	Description
Value length	12-15	The length of the content of this TLLV block in bytes, 0x00000020 (decimal 32).
Initialization vector (IV)	16-31	A 16-byte CBC initialization vector used in AES encryption and decryption of audio and video assets.
Content key (CK)	32-47	The 16-byte content key encrypted using the SK.
Padding	48- <i>n</i>	Random values that fill out the TLLV to a multiple of 16 bytes. See the description of the <a href="#">Padding</a> field in Table 2-2.

### Returning SPC Blocks in the CKC Payload

The SPC payload contains a tag return request. This TLLV contains a list of the tags of other TLLVs. The KSM must return those TLLVs (unaltered, with original tag and contents) at the end of the payload of the CKC.

### Encrypting the CKC Payload

The CKC blocks that the KSM encrypted form the CKC payload. The blocks include the following, in random order:

- An encrypted content key TLLV block containing the 16-byte encrypted content key, as described in [Encrypting the Content Key](#).
- The R1 TLLV block from the SPC payload; see [Session Key and Random Value Block](#).
- All TLLV blocks from the SPC payload that must be returned in the CKC payload, as described in [Returning SPC Blocks in the CKC Payload](#).

To encrypt the CKC payload, compute the AR\_key value by taking the first 16 bytes of an SHA-1 digest of the R1 value sent in the payload of the SPC; see [Session Key and Random Value Integrity Block](#). That AR\_key value is then used as a key to encrypt the AR seed obtained from the SPC payload (see Table 2-4).

The resulting encrypted AR seed is the key that encrypts the CKC data section using AES-128 with cipher block chaining (CBC). The KSM generates the CKC data initialization vector, sent to the Apple device in bytes 8-23 of the CKC message, shown in Table 2-10.

In cryptographic formula syntax, encrypting the payload of the CKC consists of the following process.

AR\_key = first 16 bytes of SHA-1(R1) where

R1 represents the content of R1 block from the SPC payload.

[AR] = AES\_ECB e(AR Seed)<sub>AR\_key</sub> where

AR Seed represents the content of AR seed block from the SPC payload.

[CKC data] = AES\_CBC<sub>IV</sub> e([CK] block, R1 block, Requested SPC blocks)<sub>[AR]</sub>  
where

IV represents the random initialization vector generated by the KSM.

[CK] block represents the TLLV block containing the encrypted content key.

R1 block represents the R1 block from the SPC payload.  
Requested SPC blocks represents the TLLV blocks listed in an SPC tag return request.  
[CKC data] represents the CKC payload.

# Understanding the TLLVs

## Capabilities

The Capabilities TLLV communicates features supported by the Apple device to KSM. The Capabilities TLLV is part of the SPC sent to the KSM.

**Table 3-1 Capabilities TLLV**

Field name	Byte range	Description
TLLV tag	0-7	An 8-byte value of 0x9c02af3253c07fb2.
Total length	8-11	The total length of this TLLV block in bytes. The amount of padding at the end of the block, if any, determines the total length. It must be a multiple of 16 and greater than 31.
Value length	12-15	The length of the content of this TLLV block in bytes, 0x00000010 (decimal 16).
Capabilities bits (high)	16-23	An 8-byte field containing capability bits 64-127. See Table 3-2.
Capabilities bits (low)	24-31	An 8-byte field containing capability bits 0-63. See Table 3-2.
Padding	32- <i>n</i> ( <i>padding_size</i> )	Random values that fill out the TLLV to a multiple of 16 bytes. See the description of the <b>Padding</b> field in Table 2-2.

The two capabilities bits fields are effectively a 128-bit long value which is split into two 64-bit values for ease of processing. Each capability bit indicates whether the Apple device supports a specific feature.

**Table 3-2 Features**

Feature	Capability bit	Description
HDCP Enforcement	0	When set, indicates that the Apple device can enforce the HDCP restrictions given in the HDCP Enforcement TLLV. When not set, the device cannot guarantee the HDCP restrictions given in the HDCP Enforcement TLLV, and therefore it is not recommended to release keys requiring HDCP Type 1 to the device. Regardless of the setting of this bit, if the HDCP Enforcement TLLV is not present, the device will enforce HDCP Type 0. See Table 3-13.
Offline Key	1	When set, indicates that the Apple device is capable of handling and enforcing the Offline key TLLV. See <a href="#">Offline Rental Support</a> .



Feature	Capability bit	Description
Secure Invalidation	2	When set, indicates that the Apple device is capable of supporting secure invalidation requests.
Offline Key TTLV v2	3	When set, indicates that the Apple device can support Offline Key TTLV version 2.
Security Level Baseline	4	When set, indicates that the Apple device can support enforcement of security level Baseline. <b>Note:</b> We recommend using the newer Security Level Report TTLV instead if sent by the client device.
Security Level Main	5	When set, indicates that the Apple device can support enforcement of security level Main. <b>Note:</b> We recommend using the newer Security Level Report TTLV instead if sent by the client device.

## Device Identity

**Table 3-3 Device Identity TLLV**

Field name	Byte range	Description
TLLV tag	0-7	An 8-byte value of <code>0x94c17cd676c69b59</code> .
Total Length	8-11	The total length of this TLLV block in bytes. The length is determined by the amount of padding at the end of the block, if any; this value must be a multiple of 16 and greater than 32.
Value Length	12-15	The length of the content of this TLLV block in bytes.
Version	16-19	TLLV version.
Device Class	20-23	For example: Apple Mobile or Partner LivingRoom (see below).
Vendor hash	24-31	8 byte value uniquely identifying device vendor.
Product hash	32-39	8 byte value uniquely identifying product.
FairPlay version REE	40-43	Version of FairPlay software running in REE/userland.
FairPlay version TEE	44-47	Version of FairPlay software running in TEE/kernel.
OS Version	48-51	OS version (Apple devices only). Concatenation of <code>00    major    minor    patch</code> .
Padding	52- <i>n</i>	Random values to fill out the TLLV to a multiple of 16 bytes. See the description of the <code>Padding</code> field in Table 2-2 of <i>FairPlay Streaming Programming Guide</i> .

Device Class:

```
enum
{
    kFPIDeviceClassUnknown      = 0,

    // Apple devices
    kFPIDeviceClassAppleLiving  = 1,
    kFPIDeviceClassAppleMobile  = 2,
    kFPIDeviceClassAppleDesktop = 3,
    kFPIDeviceClassAppleSpatial = 4,
    kFPIDeviceClassAppleUnknown = 127,
```

```
// Partner devices
kFPDIDeviceClassPartnerLiving  = 128,
kFPDIDeviceClassPartnerUnknown = 255,

kFPDIDeviceClassMax = 255,
};
```

## Device Info Support

FPS supports a Device Info TLLV in the SPC that reports the Apple device type and OS version.

**Note:** We recommend using the newer Device Identity TLLV instead if sent by the client device.

**Table 3-4 Device info TLLV**

Field name	Byte range	Description
TLLV tag	0-7	An 8-byte value of <code>0xd43fc6abc596aae7</code> .
Total length	8-11	The total length of this TLLV block in bytes. The amount of padding at the end of the block, if any, determines this length. It must be a multiple of 16 and greater than 31.
Value length	12-15	The length of the content of this TLLV block in bytes.
Apple device type	16-23	See Table 3-5.
OS version	24-27	Concatenation of <code>00    major    minor    patch</code> .
Version	28-31	TLLV version. Currently supported version is 1.
Padding	32- <i>n</i> ( <i>padding_size</i> )	Random values that fill out the TLLV to a multiple of 16 bytes. See the description of the <code>Padding</code> field in Table 2-2.

**Table 3-5 Device type**

Device type field	Apple device type
<code>"0x358c41b1ec78f599"</code>	Mac
<code>"0xc1500767c86c1fae"</code>	AppleTV, FPS-enabled TV or STB
<code>"0x8551fd5e31f479b3"</code>	iPhone, iPad, iPod
<code>"0x5da86ac0c57155dc"</code>	Apple Watch

# Kext Deny List

macOS devices report the version of current “Kext Deny List” (KDL) loaded on the Apple device. If the reported version is less than the latest published version, the server should treat the Apple device Security Level as AppleBaseline.

As of September 2024 the latest KDL version is 31.

**Table 3-6 Kext Deny List TLLV (included in the SPC sent to the server)**

Field name	Byte range	Description
TLLV tag	0-7	An 8-byte value of 0x70eca6573388e329.
Total length	8-11	The total length of this TLLV block in bytes. The amount of padding at the end of the block, if any, determines this length. It must be a multiple of 16 and greater than 31.
Value length	12-15	The length of the content of this TLLV block in bytes.
KDL version	16-19	Kext Deny List version.
Padding	20- <i>n</i> ( <i>padding_size</i> )	Random values that fill out the TLLV to a multiple of 16 bytes.

## Security Level Support

The Security Level of an Apple device provides information about the security robustness level of the Apple device.

The SPC reports the Security Level of the Apple device in the Security Level Report TLLV. The Key Server Module can enforce a policy based on this. For example, the Key Server Module can refuse to deliver high-value assets (4K / HDR) to AppleBaseline or Baseline devices. See Table 3-7 and Table 3-8.

Using the optional Required Security Level TLLV, the Key Server Module may indicate the minimum required Security Level of the client device to allow playback of the requested asset.

AppleBaseline/Baseline Platforms will not play video content restricted to AppleMain/Main devices.

AppleBaseline, Baseline, AppleMain and Main Security Level should only be used for video content. Audio content must use Audio Security Level.

These are further described in Table 3-9, Table 3-10, and Table 3-11.

**Table 3-7 Security Level Report TLLV (included in the SPC sent to the server)**

Field name	Byte range	Description
TLLV tag	0-7	An 8-byte value of <code>0xb18ee16ea50f6c02</code> .
Total length	8-11	The total length of this TLLV block in bytes. The amount of padding at the end of the block, if any, determines this length. It must be a multiple of 16 and greater than 31.
Value length	12-15	The length of the content of this TLLV block in bytes.
Version	16-19	TLLV version. Currently supported version is 1.
Reserved	20-23	This field is reserved
Security Level	24-31	Security Level of the Apple device. See Table 3-8.
KDL version	32-35	Kext Deny List version. If the Kext Deny List TLLV is present, the value should be used from that TLLV instead.
Padding	36- <i>n</i> ( <i>padding_size</i> )	Random values that fill out the TLLV to a multiple of 16 bytes

**Table 3-8 Security Level**

Name	Value	Notes
AppleBaseline/Baseline	"0x32f0004966a5c4f8"	Any platform that supports FairPlay streaming.
AppleMain/Main	"0x4e7fd92421d588b4"	Any platform that supports FairPlay Streaming and guarantees enhanced content protection robustness (sufficient for studio 4K / HDR playback).

**Table 3-9 Required Security Level TLLV (included in the CKC sent by the server)**

Field name	Byte range	Description
TLLV tag	0-7	An 8-byte value of 0x644cb1dac0313250.
Total length	8-11	The total length of this TLLV block in bytes. The amount of padding at the end of the block, if any, determines this length. It must be a multiple of 16 and greater than 31.
Value length	12-15	The length of the content of this TLLV block in bytes.
Version	16-19	TLLV version. Currently supported version is 1.
Reserved	20-23	This field is reserved and should be set to 0.
Security Level	24-31	Security Level. See Table 3-10.
Padding	32- <i>n</i> ( <i>padding_size</i> )	Random values that fill out the TLLV to a multiple of 16 bytes.

**Table 3-10**

Name	Value
Audio	"0x17d99d574eed567d"
AppleBaseline/Baseline	"0x32f0004966a5c4f8"
AppleMain/Main	"0x4e7fd92421d588b4"

**Table 3-11 FPS Error Messages Content-Protection violation**

Message	Description
---------	-------------

-42811	The FPS library returns this error code when there is a Security Level violation.
--------	---



# HDCP Enforcement

High bandwidth Digital Content Protection (HDCP) is a digital rights protection method that provides a secure connection between the source and the display by encrypting the audio/video stream to prevent illegal copying of the content. Using the optional HDCP Enforcement TLLV, the requirement and version of HDCP may be defined.

Table 3-12 HDCP Enforcement TLLV

Field name	Byte range	Description
TLLV tag	0-7	An 8-byte value of 0x2e52f1530d8ddb4a.
Total Length	8-11	The total length of this TLLV block in bytes. The amount of padding at the end of the block determines the total length, which must be a multiple of 16 and greater than 32.
Value Length	12-15	The length of the content of this TLLV block in bytes, 0x00000010 (decimal 16).
HDCP requirement	16-31	A 16-byte field containing the HDCP level values. See Table 3-13.
Padding	32-n (padding_size)	Random values to fill out the TLLV to a multiple of 16 bytes. See the description of the Padding field in Table 2-2.

Table 3-13 HDCP level values

Value	Description
0xEF72894CA7895B78	HDCP not required.
0x40791AC78BD5C571	HDCP Type 0 is required.
0x285A0863BBA8E1D3	HDCP Type 1 is required.

## Virtual Machine Support

If a FairPlay license is requested from an Apple device running inside a Virtual Machine, the SPC will contain a VM Device Information TLLV so that the Key Server Module can decide whether or not to issue the license.

**Table 3-14 VM Device Information TLLV**

Field name	Byte range	Description
TLLV tag	0-7	An 8-byte value of <code>0x756440e240499f70</code> .
Total Length	8-11	The total length of this TLLV block in bytes. The amount of padding at the end of the block determines the total length, which must be a multiple of 16 and greater than 32.
Value Length	12-15	The length of the content of this TLLV block in bytes.
Version	16-19	TLLV version. Currently supported version is 1. A value of 0 indicates the requesting device is not running in a virtual machine.
Host Device Class	20-23	Device class of the VM host. See <a href="#">Device Identity TLLV</a> for values.
Host OS Version	24-27	OS version of the VM host. Concatenation of: <code>00    major    minor    patch</code>
Host Virtualization Protocol Version	28-31	FairPlay virtualization protocol version used by the VM host.
Guest Device Class	32-35	Device class of the VM guest. See <a href="#">Device Identity TLLV</a> for values.
Guest OS Version	36-39	OS version of the VM guest. Concatenation of: <code>00    major    minor    patch</code>
Guest Virtualization Protocol Version	40-43	FairPlay virtualization protocol version used by the VM guest.
Padding	44- <i>n</i> ( <i>padding_size</i> )	Random values to fill out the TLLV to a multiple of 16 bytes. See the description of the <code>Padding</code> field in Table 2-2.

# Renting and Leasing the Content Key

FPS supports time-sensitive content keys. Renting and leasing set specific limits on an app’s access to FPS decryption keys. The server may associate a rental period with the media content and/or a lease period with the Apple device. The server’s CKC response contains the validity duration of the content key.

## Content Key Expiration

FPS’s content key expiration creates two modes of time-sensitive exchange: video rental and secure lease. These modes are used separately or together.

### Video Rental

The content key is a rental type. FPS does not start the decryption if the content key has expired. However, FPS continues the user experience if the content key expires during the playback. When started again with an expired key, the Apple device declines the playback.

### Secure Lease

The content key is a lease type. Typically, a content provider policy would restrict the number of simultaneous playbacks (slots) for a user account. The server associates a slot to a device, and the server delivers the content key with the expiration that represents the lease. The Apple device may request that the key be renewed by the server before the lease expires. The server provides a new expiration time for the content key, and playback continues uninterrupted. If the content key is not renewed, the Apple device stops the playback when the lease expires. The server recognizes that playback has stopped and frees the device slot.

This design ensures that a device is not orphaned (in a stale state) based on time rather than messaging and garbage collection. The expiration triggers a server event to securely release the device slot. The server knows playback stopped and frees the device slot as soon as the content key expires and the PlayContent is discarded. Using the secure lease and the device identification, the server can implement a robust solution for the management of simultaneous streams maintaining a seamless user experience.

## TLLV for Rental and Lease

The SPC includes a specific TLLV to provide the state of the media content playback. The key server uses this TLLV to manage the rental period and the lease period. Details of the media playback state TLLV are in Table 4-1.

**Table 4-1 Media playback state TLLV**

Field name	Byte range	Description
TLLV tag	0-7	An 8-byte value of 0xeb8efdf2b25ab3a0.
Total Length	8-11	The total length of this TLLV block in bytes. The total length is determined by the amount of padding at the end of the block, if any; it must be a multiple of 16 and greater than 32.

Field name	Byte range	Description
Value Length	12-15	The length of the content of this TLLV block in bytes, 0x00000010 (decimal 16).
Creation Date	16-19	The time in seconds from Jan 1, 1970 to the time when the SPC was created.
Playback State	20-23	The playback state of the Apple device at the time the SPC was created. Possible values are listed in Table 4-2.
Session ID	24-31	An ID that represents the playback of a media content independently of its bit rates and content keys. When the user closes and re-opens a movie, the Apple device generates a new Session ID to identify the new instance.
Padding	32- <i>n</i> ( <i>padding_size</i> )	Random values to fill out the TLLV to a multiple of 16 bytes. See the description of the <code>Padding</code> field in Table 2-2.

**Table 4-2 Apple device playback states**

TLLV field value	Playback state
0xf4dee5a2	State 1: The Apple device is ready to start playing. The response CKC must contain a valid content key.
0xa5d6739e	State 2: The playback stream is playing or paused. The KSM must reply with a CKC containing a rent/lease response TLLV, but it does not need to contain a valid content key.
0x4f834330	State 3: The playback stream is playing, but the lease is about to expire. The response CKC must contain a valid content key.

## Establishing the Rental and Lease Period

When a KSM receives an SPC with a media playback state TLLV, the KSM may include a content key duration TLLV in the CKC message that it returns. If the Apple device finds this type of TLLV in a CKC that delivers an FPS content key, it will honor the terms of the rental or lease or both when the key is used. Table 4-3 lists the fields of the rental and lease response TLLV.

**Note:** The app is unable to modify or overrule the rental and lease periods specified in the CKC.

**Table 4-3 Content key duration TLLV**

Field name	Byte range	Description
TLLV tag	0-7	An 8-byte value of 0x47acf6a418cd091a.
Total Length	8-11	The total length of this TLLV block in bytes. The amount of padding at the end of the block, if any, determines this length. this value must be a multiple of 16 and greater than 32.
Value Length	12-15	The length of the content of this TLLV block in bytes, 0x00000010 (decimal 16).
Lease Duration	16-19	The duration of the lease, if any, in seconds.
Rental Duration	20-23	The duration of the rental, if any, in seconds.
Key Type	24-27	The key type. Possible values are listed in Table 4-4.
Reserved	28-31	Reserved; set to a fixed value of 0x86d34a3a.
Padding	32- <i>n</i> ( <i>padding_size</i> )	Random values to fill out the TLLV to a multiple of 16 bytes. See the description of the <code>Padding</code> field in Table 2-2.

**Table 4-4 Rental and lease key types**

TLLV field value	Type of rental or lease
0x1a4bde7e	Content key valid for lease only.
0x3dfe45a0	Content key valid for rental only.
0x27b59bde	Content key valid for both lease and rental.

See [Offline FairPlay Streaming](#) for additional rental and lease key types to support persistent keys.

## Simultaneous Renting and Leasing

It is possible to combine a rental and a lease into one CKC. The combination of renting and leasing follows these general rules:

- A rental period covers the initial delivery of a content key and the start of a stream. If the rental period expires during playback, the stream continues to play until the media playback stops.
- A lease period covers the validity of the content key for media playback. If the lease period expires during playback, the media playback stops.
- In a combined renting and leasing arrangement, the mechanism by which leasing registers the playback device may be used to limit the rental to that device only, as a leasing restriction enforced by the KSM.

- If the lease expires before the end of the rental period, the key server should allow the lease to be renewed.

# Offline FairPlay Streaming

**Note:** Offline FairPlay Streaming (FPS) is an extension of offline HTTP Live Streaming (HLS). This guide describes how to build on top of the technologies to achieve an Offline FairPlay Streaming solution. The term “HLS” is used to describe technology that is common to both FPS and HLS.

Apps can save HTTP Live Streaming assets onto Apple devices. This is known as Offline HLS. This new capability allows users to download and store their HLS movies while they have access to a fast, reliable network, and watch them later without a network connection.

Offline HLS is supported starting with the following OS versions:

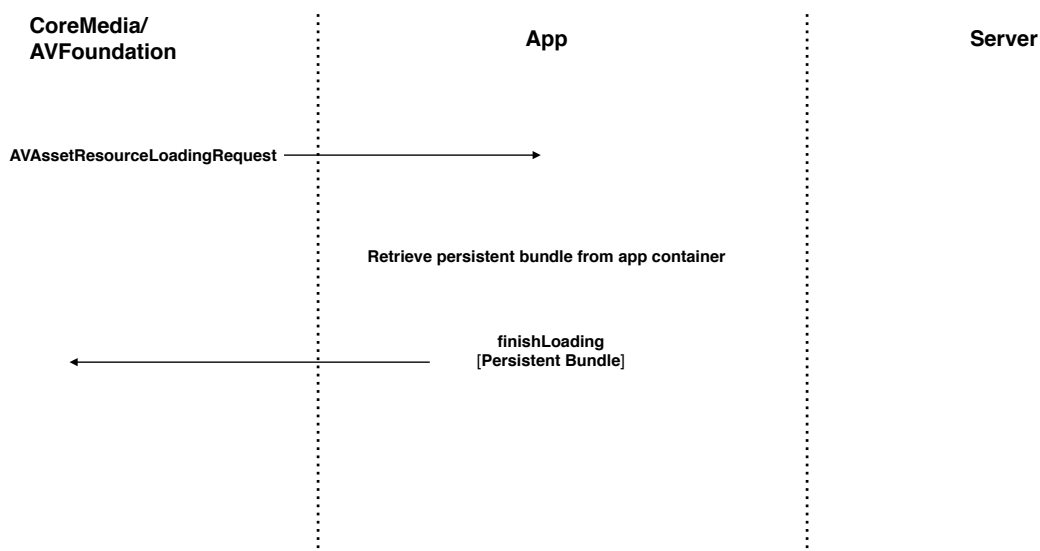
- iOS: 10.0
- macOS: 10.15
- iPadOS: all versions

Offline HLS assets can be SAMPLE-AES encrypted. There are a few additional steps required for downloading and managing encrypted Offline HLS assets to ensure the playability of downloaded assets when no network connection is present.

When creating an `AVURLAsset` for use in downloading an Offline HLS asset, apps must install a delegate to handle encryption keys. The persistent keys are not stored with downloaded HLS assets. Instead, apps must store and manage persistent keys, using an `AVAssetResourceLoader` and a delegate object implementing the `AVAssetResourceLoaderDelegate` protocol. The server may enable an Apple device to persist the key either indefinitely or for the provided validity duration. To enable persistence of the content key, the server’s CKC response shall contain a Content key duration TLLV (TLLV tag `0x47acf6a418cd091a`). See Table 4-3.

**Figure 5-1** Offline playback

## Offline Playback with Persistent Bundle



### Preloading Offline Keys

Because an `AVAssetDownloadTask` can start while your app is background suspended, the recommendation is to preload any content keys using `– [AVContentKeySession processContentKeyRequestWithIdentifier:initializationData:options:]` or `AVAssetResourceLoader.preloadsEligibleContentKeys`. However, AVFoundation will attempt to load any resources requiring an `AVContentKeySessionDelegate` or `AVAssetResourceLoadingDelegate` (HLS playlists with custom URL schemes and FPS keys) while your app is still running. The Apple device must ensure that all the keys have been loaded before starting a background download task on the `AVURLAsset`.

The final policy decision as to which content keys can be persisted on the device belongs to the Key Server Module vending the CKC. The server has the option to allow the Apple device to persist the key either indefinitely, or for a specified duration. To enable persistence of the content key, the server's CKC response shall contain a Content key duration TLLV.

FairPlay Streaming on the Apple device does not start the decryption if the persisted content key has expired. However, playback on the Apple device continues even if the content key expires during playback.

**Table 5-1** Rental and lease key types for persistence

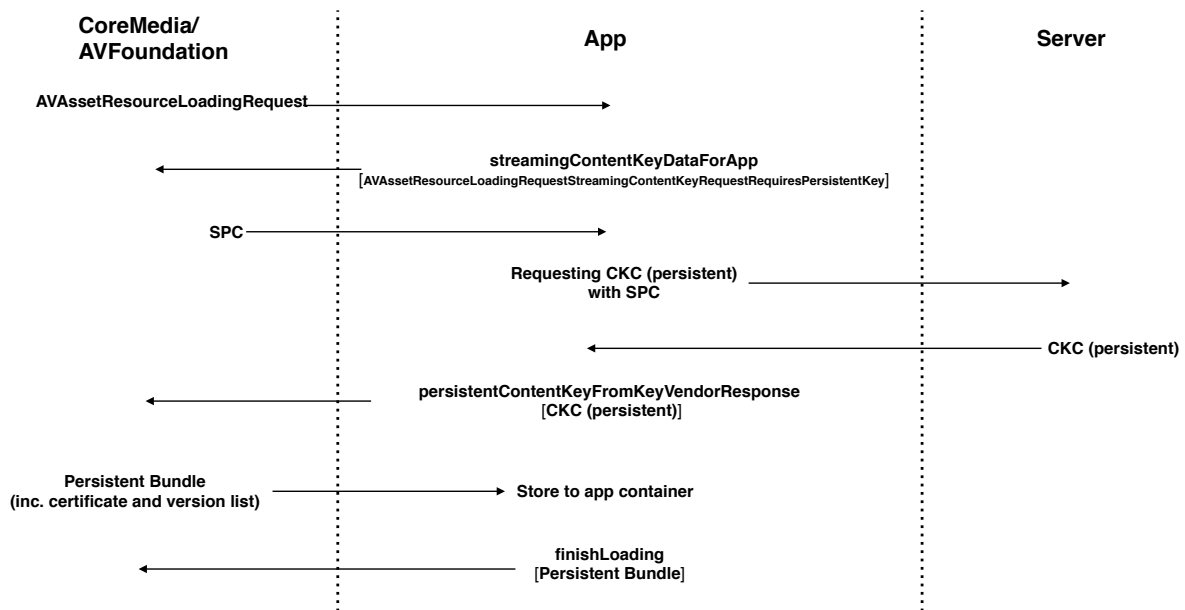
TLLV field value	Type of rental or lease
0x3df2d9fb	Content key can be persisted with unlimited validity duration.
0x18f06048	Content key can be persisted, and its validity duration is limited to the “Rental Duration” value.



The diagram below shows the life cycle of a persistent key request.

**Figure 5-2** Persistent key request

## Persistent Key Request



**Table 6-2** FPS error messages for presistence

Message	Description
-42799	FPS returns this error code when Apple issues a security update and the existing persistent key format is no longer supported. In this case, the application must request a new persistent key from the server.
-42800	FPS returns this error code when persistent key has expired. The server always has an option to add a validity period for each key it issues for offline playback. Once the validity period is over, the Apple device will refuse to decrypt offline content protected with such key and indicate the error with this error code. It is up to the application developers to decide whether to request a new key from the server or treat this error condition as a permanent expiry; for example, a recorded sporting event must not be playable after 48 hours and no key renewal is possible. To use the best practice and achieve maximum flexibility, always send a key-renewal request to the server and let the server decide whether to allow the renewal.



# Offline Rental Support

Starting with iOS 11.0, FPS supports a new feature to enable offline rental. This feature allows the server to specify two expiration times for content. This “dual expiry” is similar to how the rental of iTunes movies functions.

In iOS 12.2 and later, FPS supports relating multiple streams belonging to the same program. Offline Key TLLV version 2 adds a new field called `Title ID`. All sub-streams for a given program should have a common `Title ID`. When the application requests the FairPlay Streaming component to create a Sync SPC or invalidate a persistent key, FPS ensures that all persistent keys identified by the same `Title ID` are invalidated. The updated Sync TLLV will include a list of all the invalidated `Stream/Content IDs`. This eliminates the need to invalidate persistent keys on a one by one basis when multiple sub-streams were downloaded for the program.

**Note:** FPS supports Offline Rental in all versions of iPadOS, and in macOS 10.15 and newer.

## Storage and Playback Expiry

The server can use the new Offline Key TLLV (see below) to specify two different duration periods for the downloaded content:

1. Storage duration (seconds). This specifies the maximum time the key stays valid prior to playback being started. Measured from license acquisition time.
2. Playback duration (seconds). This specifies the maximum time the key stays valid after playback has been started. Measured from the first playback start time.

After content is downloaded and the content license is acquired, the app may report to the server the remaining key validity duration using a Sync SPC (see below). This allows the server and the client to be synchronized on when the license to play the downloaded asset will expire.

**Example:** A user rents a movie; the server sets the storage duration to 2,592,000 (30 days) and the playback duration to 86,400 (24 hours). The user has up to 30 days to start watching the movie and 24 hours to finish watching it after starting the playback. If the app requests a Sync SPC to be created prior to user starting playback, the “Duration to expiry” field of the Sync TLLV will be set to 2,592,000 minus the number of seconds passed since license was downloaded. The same SPC requested after playback has started will contain a Sync TLLV with the “Duration to expiry” field set to 86,400 minus the number of seconds since playback was started.

## Offline Key

Table 6-1 Offline Key TLLV

Field name	Byte range	Description
TLLV tag	0-7	An 8-byte value of 0x6375d9727060218c.

Field name	Byte range	Description
Total length	8-11	The total length of this TLLV block in bytes. The total length is determined by the amount of padding at the end of the block, if any; it must be a multiple of 16 and greater than 31.
Value length	12-15	The length of the content of this TLLV block in bytes.
Version	16-19	TLLV version. Currently supported version is 2. Supported in iOS 12.2 and higher.
Reserved	20-23	This field is reserved and must be set to 0.
Content ID (Stream ID in version2)	24-39	Unique content ID of the downloaded asset assigned by the server. This server receives this value in a Sync TLLV.
Storage duration	40-43	Asset storage validity duration in seconds. Starts at license acquisition time. A value of zero means no limit.
Playback duration	44-47	Asset playback validity duration in seconds. Starts at asset first playback time. A value of zero means no limit.
Title ID	48-63	(Version 2 only) Unique ID common to all streams for the five content. The server receives this value in a Sync TLLV.
Padding	64- <i>n</i> ( <i>padding_size</i> )  The byte range for version 1 is 48- <i>n</i>	Random values that fill out the TLLV to a multiple of 16 bytes. See the description of the <code>Padding</code> field in Table 2-2.

To enable dual expiry for an asset, the KSM should add an Offline Key TLLV to the returned CKC.

**Note:** An Offline Key TLLV cannot be used in the same CKC payload as an already existing Content Key Duration TLLV. If both of these TLLVs are in the CKC, the processing of the CKC stops and an Invalid CKC error is returned.

Since the CKC cannot be interpreted by the app, the server should notify the app that the asset is using dual expiry so that the app knows that a Sync SPC can be generated.

## Sync

An app can obtain a Sync SPC via the method `[AVContentKeySession makeSecureTokenForExpirationDateOfPersistableContentKey:completionHandler:]`.

This method will fail unless the persistable content key was constructed from a CKC that included an Offline Key TLLV.

The app should send the resulting SPC to the KSM. The KSM should distinguish between Sync SPCs and other SPCs. Any SPC may contain a Sync TLLV, so you need to check the Sync TLLV for validity. Valid Sync

TLLVs will have the Version field set to 1 or 2. Any other value is invalid. A Sync SPC is one with a valid Sync TLLV.

**Table 6-2 Sync TLLV (version 1)**

Field name	Byte range	Description
TLLV tag	0-7	An 8-byte value of 0x77966de1dc1083ad.
Total length	8-11	The total length of this TLLV block in bytes. The amount of padding at the end of the block, if any, determines this length. It must be a multiple of 16 and greater than 31.
Value length	12-15	The length of the content of this TLLV block in bytes.
Version	16-19	TLLV version. Currently supported version is 1.
Reserved	20-23	This field is reserved and must be set to 0.
Content ID	24-39	Unique content ID of the downloaded asset received in Offline Key TLLV.
Duration to expiry	40-43	Remaining license validity time in seconds. It will be set to 0 if the license has expired, and to 0xFFFFFFFF if the license doesn't have an expiry date.
Padding	44- <i>n</i> ( <i>padding_size</i> )	Random values that fill out the TLLV to a multiple of 16 bytes. See the description of the <code>Padding</code> field in Table 2-2.

**Table 6-3 Sync TLLV (version 2)**

Field name	Byte range	Description
TLLV tag	0-7	An 8-byte value of 0x77966de1dc1083ad.
Total length	8-11	The total length of this TLLV block in bytes. The amount of padding at the end of the block, if any, determines this length. It must be a multiple of 16 and greater than 31.
Value length	12-15	The length of the content of this TLLV block in bytes.
Version	16-19	TLLV version. For Apple devices supporting Offline Rental, the version is set to 2.
Reserved	20-23	This field is reserved and must be set to 0.
Server Challenge	24-31	The unique 64-bit server challenge generated by the server.

Field name	Byte range	Description
Flags	32-39	64-bit integer containing the current flag setting. See the flag values in Table 6-4.
Title ID	40-55	The 128 bit title ID provided in the Offline Key TLLV.
Duration to expiry	56-59	Remaining license validity time in seconds. It will be set to 0 if the license has expired, and to 0xFFFFFFFF if the license doesn't have an expiry date.
Records Invalidated	60-63	The total number of invalidated records.
Invalidated Stream IDs	64-X	A concatenated array of invalidated Stream IDs.
Padding	X+1-n (padding_size)	Random values that fill out the TLLV to a multiple of 16 bytes. See the description of the <code>Padding</code> field in Table 2-2.

The following table contains the possible values for the `Flag` field:


**Table 6-4 Sync TLLV Flag Field Definitions**

Flag	Capability Bit	Description
KD_SYNC_SPC_FLAG_REPORT	0	The Apple device requested a sync.
KD_SYNC_SPC_FLAG_SECURE_INVALIDATION	1	The Apple device requested a secure invalidation.
KD_SYNC_SPC_FLAG_SECURE_INVALIDATION_ALL	2	The Apple device requested a "Delete all" operation.
KD_SYNC_SPC_FLAG_SUCCESS	16	The requested operation was successful.
KD_SYNC_SPC_FLAG_OBJ_NOT_FOUND	17	The provided persistent key was not found or is invalid.
KD_SYNC_SPC_FLAG_OBJ_EXPIRED	18	The provided persistent key is valid, but expired by the time of the request.

# Developing an FPS-Aware App

Any media playback app that runs on an Apple device can implement FPS. This chapter covers programming required for an app to obtain content keys that decrypt FPS media.

Typically, playback apps provide a user interface for browsing and selecting the content to be streamed, support user identification, and facilitate other user and content provider transactions. Additionally, an FPS-aware playback app must establish two-way communication between the Apple device and a key server to support FPS functions.

 **Warning:** FPS cannot be run on iOS Simulator.

For general information about writing apps for Apple iOS devices, visit the [Developer Center](#).

## Identifying Your FPS App with an Application Certificate

As part of registering an FPS playback app, you provide Apple with an X.509 Certificate Signing Request linked to your private key. In return, you receive an Application Certificate encoded with the X.509 standard with distinguished encoding rules (DER). Bytes 152-171 of the SPC message contain a secure hash algorithm (SHA-1) digest of that encoded certificate.

Every playback app that uses FPS must find the media's key server and establish communication with that server. When the Apple device and the key server can exchange messages, the app must send the server an FPS-created SPC message. This message contains a hash of the Application Certificate identifying your private key.

The recommendations below help you ensure FPS security.

- Do not hard-code the Application Certificate in the playback application.
- Verify that the hash value in bytes 152-171 of the SPC correctly identifies the private key of the developer from which the module expects to receive SPC messages.
- Do not enforce the expiration date of the Application Certificate within your app. FPS does not enforce the expiration date.

In the code sample shown in the iOS FPS Client sample (included in the SDK), `kTestAppCert` contains the Application Certificate.

## Integrating FPS with the iOS Decryption Process

To use FPS, the playback app must implement the `AVAssetResourceLoaderDelegate` protocol. For each `AVURLAsset` subclass required by FPS, the app must create an appropriate object that implements this protocol as the `AVAssetResourceLoader` delegate for that subclass. `AVAssetResourceLoader` invokes this delegate to examine URL requests that the operating system cannot handle by itself, including requests for content keys.

The app uses the `resourceLoader` property of the `AVURLAsset` subclass to obtain the instance of `AVAssetResourceLoader` associated with the class. It uses the `AVAssetResourceLoader` method

`–setDelegate:queue:` to set the delegate and the dispatch queue on which `AVAssetResourceLoader` will invoke the delegate.

## Integrating FPS in Safari

The FPS content that you author for iOS and Apple TV also plays in Safari starting on macOS 10.10.3, iOS 11.2, and iPadOS. Encrypted Media Extensions (EME) provide FPS support on Safari. Support for the WebKit-prefixed EME specification (<https://dvcs.w3.org/hg/html-media/raw-file/tip/encrypted-media/encrypted-media.html>) started on macOS 10.10.3, iOS 11.3, and iPadOS. Support for the Modern EME specification (<https://w3c.github.io/encrypted-media/>) started on iOS 12.2, and macOS 10.14.4 and greater. The EME extended `HTMLMediaElement` APIs manage the FPS process with secure message exchanges analogous to FPS on iOS devices and Apple TV.

### EME Message Exchange

For web pages in Safari, FPS supports a *Key System* identified by the string `com.apple.fps`.

You must create a Key Session to provide a context for message exchange with the Content Decryption Module (CDM)/Key System. Per the EME specification, each Key Session is associated with a single instance of *Initialization Data* provided in the `createSession()` call.

For FPS, this Initialization Data must be the following byte array.

`AssetID + Certificate`

In this expression, `AssetId` represents the byte array defined in Table 2-4, `Certificate` represents the Application Certificate provided by Apple, and the `+` indicates concatenation of the two values. The `AssetId` can be any string you choose.

Use the following events in your JavaScript for Safari to support FPS.

#### `encrypted`

The `encrypted` event finds the CDM, identified with the string `com.apple.fps` and allows for creation of the `keySession`. The event triggers when a process requests playback of FPS protected content.

#### `message`

The `message` event sends the SPC and obtains a CKC from the Key Server Module. The `update()` function adds the CKC to the `keySession`.

#### `keystatuseschange`

The `keystatuseschange` event triggers when a change occurred in the keys in the session or their status. See <https://w3c.github.io/encrypted-media/#mediakeystatusmap-interface> for more information.

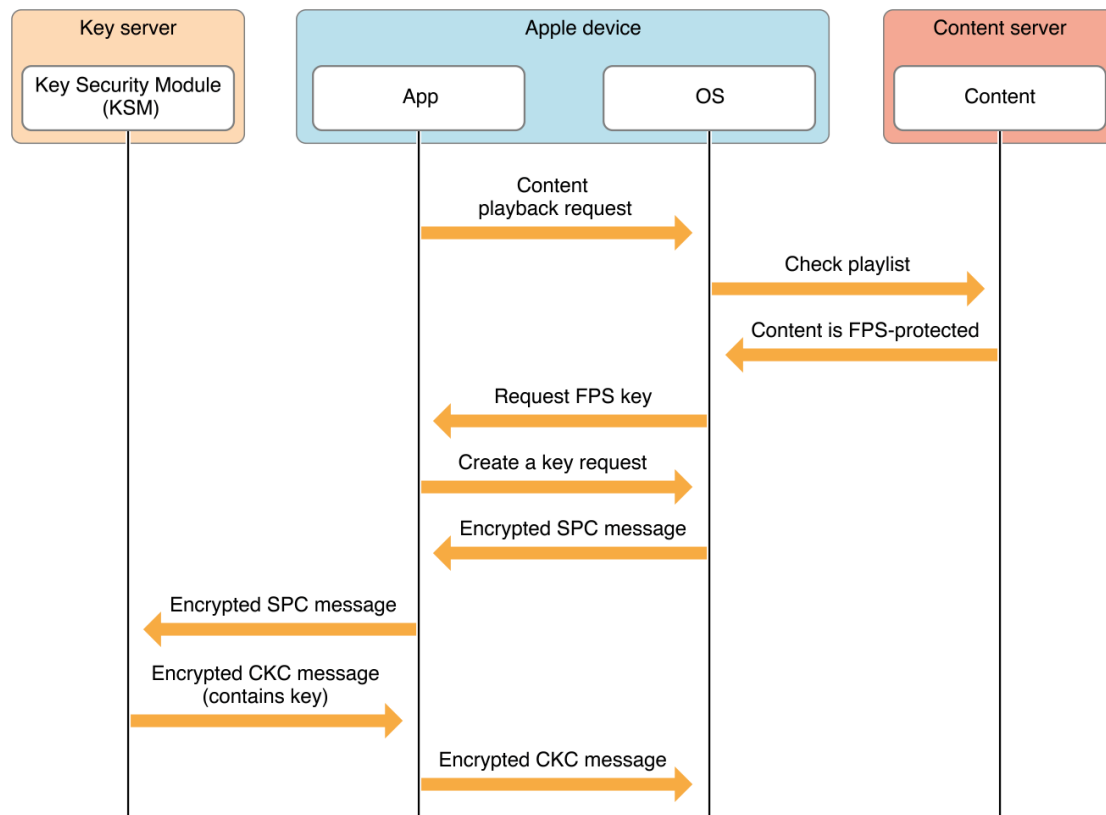
Apple provides samples in the FPS SDK, including a JavaScript implementation of the API for Safari on macOS.



## Requesting a Content Key from the Key Server

When the operating system asks the app to provide a content key, as shown in Figure 8-1, the app invokes the `AVAssetResourceLoader` delegate's implementation of its `-resourceLoader:shouldWaitForLoadingOfRequestedResource:` method. This method provides the delegate with an instance of `AVAssetResourceLoadingRequest`, which accesses the underlying `NSURLRequest` for the requested resource and support for responding to the request.

**Figure 8-1** FPS information flow



When the request is for a content key, the app invokes the delegate – `[AVAssetResourceLoadingRequest streamingContentKeyRequestDataForApp:contentIdentifier:options:error:]` method. This method obtains the SPC message from the operating system. Then the app sends the SPC to the key server, as shown in Figure 8-1, using appropriate transport forms and protocols.

## Processing the Key Server's Response

The KSM constructs the CKC message containing the content key as described in [Programming the Key Server Module](#). The key server returns a CKC message in response to the app's SPC message, as shown in Figure 8-1. After receiving this message, the app sends it to the operating system by invoking the `AVAssetResourceLoadingRequest` method – `[AVAssetResourceLoadingRequest finishLoading]`. The device can now decrypt and play the content stream using HLS as summarized in [HTTP Live Streaming Overview](#).

# Configuring AirPlay Mode

When an Apple device is in AirPlay mode, FPS content will not play on an attached Apple TV unless AirPlay playback is set to mirroring. The FPS-aware app must set the `usesExternalPlaybackWhileExternalScreenIsActive` property of the `AVPlayer` object to `TRUE` with code such as this:

```
// create AVPlayer object
player = [AVPlayer playerWithURL:movieURL];
// set the property to TRUE
player.usesExternalPlaybackWhileExternalScreenIsActive = TRUE;
```

# Interpreting Error Messages

When `-streamingContentKeyRequestDataForApp:contentIdentifier:error:` fails, it returns `nil` and sets the `outError` parameter to an instance of `NSError` that describes the failure. In this case, invoke `-finishLoadingWithError:`, passing the resulting error.

If you want the application to report the error to the user at this stage of the process, use the `localizedDescription` of `NSError` from `-streamingContentKeyRequestDataForApp:contentIdentifier:error:`. You can access another error instance through the `NSUnderlyingErrorKey` in the `userInfo` dictionary of the `NSError` instance provided by `-streamingContentKeyRequestDataForApp:contentIdentifier:error:`. This `NSError` instance provides additional information of interest when debugging an application that tries to obtain an SPC. It can contain one of the codes listed in Table 8-1.

**Table 8-1 FPS error messages**

Message	Description
-42656	Lease duration has expired.
-42668	The CKC passed in for processing is not valid.
-42672	A certificate is not supplied when creating SPC.
-42673	<code>assetId</code> is not supplied when creating an SPC.
-42674	Version list is not supplied when creating an SPC.
-42675	The <code>assetID</code> supplied to SPC creation is not valid.
-42676	An error occurred during SPC creation.
-42679	The certificate supplied for SPC creation is not valid.
-42681	The version list supplied to SPC creation is not valid.
-42783	The certificate supplied for SPC is not valid and is possibly revoked.

Message	Description
-42803	Offline key is invalid.

## Manually Fetching FPS Error Messages

The following code fetches an underlying error specific to FPS.

### Listing 8-1 Manually fetching FPS errors

```

NSError *topLevelError = nil;
NSData *requestData = [loadingRequest
streamingContentKeyRequestDataForApp:appID contentIdentifier:contentID
error:&topLevelError];
if (requestData == nil && topLevelError != nil)
{
    NSError *underlyingError = [[topLevelError userInfo] objectForKey:
    NSUnderlyingErrorKey];
    if ([[underlyingError domain] isEqualToString:NSOSStatusErrorDomain])
    {
        NSInteger errorCode = [underlyingError code];
        // check whether this errorCode is specific to FPS as listed in
        Table 3-1.
    }
}

```

# Formatting and Encrypting Streams

You've seen how the KSM works and how to build an app to communicate with that KSM. Now that you understand some aspects of FPS on the key server and the Apple device, take a look at the content that streams between the two. FPS requires content formatting and encrypting in accordance with the HTTP Live Streaming (HLS) and MPEG-2 stream encryption standards published in [HTTP Live Streaming IETF draft](#) and [MPEG-2 Stream Encryption Format for HTTP Live Streaming](#). This chapter helps you prepare your FPS content for these standards.

Beyond this chapter, the following books provide instruction on formatting and encrypting streams.

- *HTTP Live Streaming IETF draft* details the media formatting required to send your content via HTTP. The advantages of HLS and implementation instructions are provided in [HTTP Live Streaming Overview](#).
- *MPEG-2 Stream Encryption Format for HTTP Live Streaming* explains features of the MPEG-2 standard.

## Preparing Content for FPS

As shown in [Example Playlist Files for use with HTTP Live Streaming](#), HLS extends the m3u playlist format with an EXT-X-KEY tag. FPS requires that this tag be included in the HLS playlist and that it declare the following attributes:

- **METHOD**: The encryption method. **SAMPLE-AES** indicates AES-128\_CBC unpadded encryption of individual samples.
- **URI**: The path for obtaining the content key. An example is `skd://key65`, as shown in Listing 9-1.
- **KEYFORMAT**: A value of `com.apple.streamingkeydelivery` indicates a FPS key; `identity` indicates the original key format of clear text 16-byte AES key.
- **KEYFORMATVERSIONS**: A list of key format versions separated by slashes. For example, `1/2` indicates support for either version 1 or version 2 of the key format.

The following listing shows a sample FPS EXT-X-KEY tag in a streaming playlist.

### Listing 9-1 Adding FPS to an HLS Playlist

```
#EXT-X-KEY:METHOD=SAMPLE-AES,URI="skd://key65",  
KEYFORMAT="com.apple.streamingkeydelivery",KEYFORMATVERSIONS="1"
```

## Including Initialization Vectors (IV) in Playlists

There are a few important considerations that apply to the IV in an FPS m3u8 playlist.

- FPS does not support IVs listed in the EXT-X-KEY tag's IV attribute in an m3u8 playlist. The FPS-aware app ignores any IV in the playlist. The Key Server Module only delivers the IV in the content key context (CKC).

- FPS does not support using IVs as media sequence numbers in a m3u8 playlist. AES encryption and decryption of audio and video assets uses the IV delivered in the CKC.
- An EXT-X-KEY tag with a KEYFORMAT of identity without an IV attribute indicates that the media sequence number should be used as the IV to decrypt a media segment. However, if you specify a KEYFORMAT of `com.apple.streamingkeydelivery` to indicate an FPS key (leaving out any IV), IVs are not used as media sequence numbers.
- For more information, see <http://tools.ietf.org/html/draft-pantos-http-live-streaming>.
- The Apple device doesn't make a key request for every segment in the playlist.
  - For example, if an EXT-X-KEY tag is in the playlist along with three segments, the Apple device requests the key just once. This means that your encryption system must encrypt the three segments with the same IV.
  - In a similar example, a playlist sequence includes the following items where the key lines are identical. In this scenario, the second key line (with `Key_0`) is not necessary because the Apple device requests the key only once.
    - `Key_0`
    - `Segment_1`
    - `Key_0`
    - `Segment_2`
- Some special circumstances do require key requests for every segment, such as when using FPS through AirPlay or Digital AV Adaptors.

## Using FPS Options with the Media File Segmenter

The following new FPS-specific features are part of the `mediafilesegmenter` tool included with the HTTP Live Streaming tools download.

- `-P` is the short form of `--streaming-key-delivery`. Either form indicates that the key file is 32 bytes long, where the first 16 bytes is the content key and the second 16 bytes is the initialization vector (IV). This option is necessary for `KEYFORMAT="com.apple.streamingkeydelivery"` streaming.
- The option `--encrypt-iv` is incompatible with FPS.
- If an existing key file is supplied to `--encrypt-key-file`, it must be 32 bytes long. The first 16 bytes hold the content key, and the second 16 bytes hold the IV. The segment `/tmp/key.bin` in Listing 9-2 represents a 32-byte file.
- The `--stream-encrypt` option is necessary if the key is to be delivered via FPS.

Listing 9-2 uses the `mediafilesegmenter` command to produce an m3u8 playlist for use with FPS.

### Listing 9-2 Media File Segmenter command

```
mediafilesegmenter --stream-encrypt --streaming-key-delivery --encrypt-key-
file=/tmp/key.bin --encrypt-key-url="skd://example/key" /tmp/source.mov
```

## Using ALLOWED-CPC in playlist to improve tier selection

With “Security Level support”, platforms can be classified as follows.

**Table 9-3 Platforms**

CPC Label	Devices that adhere to that CPC level
AppleBaseline	Any Apple device that supports FairPlay Streaming.
AppleMain	Any Apple device that supports FairPlay Streaming and guarantees enhanced content protection robustness (sufficient for studio 4K / HDR playback).
Baseline	Any non-Apple device that supports FairPlay Streaming. For example, any AirPlay 2-enabled smart TV.
Main	Any non-Apple device that supports FairPlay Streaming and guarantees enhanced content protection robustness (sufficient for studio 4K / HDR playback).

The optional ALLOWED-CPC attribute of the EXT-X-STREAM-INF tag indicates which Security Level the stream requires. Leverage it to avoid requesting assets that the device will not be able to play because it does not have the required Security Level.

See Listing 9-4 for an example that uses an ALLOWED-CPC attribute.

**Listing 9-4** Adding ALLOWED-CPC to an HLS Playlist

```
#EXT-X-STREAM-INF:AVERAGE-BANDWIDTH=2266124,_AVG-  
BANDWIDTH=2266124,BANDWIDTH=3752516,VIDEO-  
RANGE=SDR,CODECS="avc1.64001f,mp4a.40.2",AUDIO="audio-stereo-160",FRAME-  
RATE=23.976,HDCP-LEVEL=TYPE-0,RESOLUTION=1186x494,ALLOWED-  
CPC="com.apple.streamingkeydelivery:AppleBaseline/Baseline"
```

**Downloading HTTP Live Streaming tools:** You must log in to the iOS developer library to access the [HTTP Live Streaming tools download](#).

# Document Revision History

This table describes the changes to *FairPlay Streaming Programming Guide*.

Date	Notes
2015-06-08	New document that describes how to implement FairPlay Streaming encryption in HTTP Live Streaming media.
2015-09-15	Added macOS support and Lease/Rental Support features. Added clarification concerning the initialization vector (IV) in an FPS m3u8 playlist and behavior of the client when making key requests for segments in a playlist.
2017-08-21	Added Capabilities TLLV and sections on Offline FairPlay Streaming and Offline Rental Support.
2019-07-11	Added enhanced Offline TLLV information to support grouping of sub-streams for a particular program.
2020-05-05	Added ALLOWED-CPC description. Added Security Level support. Added Kext Deny List support. Added Device Info support.
2020-11-05	Added intra-document links and performed a copy edit.
2021-02-08	Updated copyright and legal information.
2021-03-10	Updated external links and renamed events in “Integrating FPS in Safari.”
2021-06-11	Fixed an incorrect reference in the CKC version description in Table 2-10.
2022-03-30	HDCP Enforcement TLLV description added in Table 3-12. Device Identity TLLV description added in Table 3-3.
2022-12-01	Updated description of Version field in Device Identity TLLV (Table 3-3).
2023-08-10	Added kFPDIDeviceClassAppleSpatial device class support in Device Identity TLLV (Table 3-3).



Date	Notes
2024-01-25	Fixed offset of Duration to expiry field in Sync TLLV (Table 6-3).
2024-10-08	<p>Added description of SPC version 2 using RSA-2048.</p> <p>Updated list of SPC TLLVs (Table 2-4).</p> <p>Added new Capabilities bit descriptions (Table 3-2).</p> <p>Added Kext Deny List field in Security Level Report TLLV (Table 3-7).</p> <p>Corrected latest KDL version from 32 to 31.</p> <p>Renamed Key Security Module to Key Server Module.</p> <p>“Using the FPS SDK and Tools” chapter migrated to Key Server Module Guide</p>
2025-05-05	Added Virtual Machine Support.



Apple Inc.  
© 2023 Apple Inc.  
All rights reserved.

AirPlay, Apple, the Apple logo, Apple TV, Apple Watch, FairPlay, iPad, iPadOS, iPhone, iPod, Mac, macOS, Safari, watchOS, and visionOS are trademarks of Apple Inc., registered in the U.S. and other countries and regions.

IOS is a trademark or registered trademark of Cisco in the U.S. and other countries and regions and is used under license.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer or device for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice. No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-branded products.

Apple Inc.  
One Apple Park Way  
Cupertino, CA 95014

**APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.**

**IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT, ERROR OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.**  
Some jurisdictions do not allow the exclusion of implied warranties or liability, so the above exclusion may not apply to you.