



Modular DRM
Working with Foreign Content Keys

version 1.2

[Purpose](#)

[Summary](#)

[Encryption](#)

[Playback](#)

[Requirements](#)

[Construction of the Widevine PSSH](#)

[PSSH Insertion](#)

[Insertion of PSSH into the content](#)

[Insertion of PSSH into the MPD](#)

[Example of PSSH insertion in MPD](#)

[Playback using foreign content key\(s\)](#)

[Using the Widevine Cloud License Service](#)

[content_key_specs.track_type](#)

[content_key_specs.key_id](#)

[Example of key_id](#)

[Case Study - Using PlayReady-generated Key ID](#)

[Problem Description](#)

[Generate a Widevine PSSH using a converted GUID](#)

[Using an alternate set of Key IDs](#)

[Appendix](#)

[Generation of a Widevine PSSH Box](#)

[Client security level \(content_key_specs.security_level\)](#)

[HDCP Output Matrix](#)

[Conversion to Widevine-compatible Key ID format](#)

© 2016 Google, Inc. All Rights Reserved. No express or implied warranties are provided for herein. All specifications are subject to change and any expected future products, features or functionality will be provided on an if and when available basis. Note that the descriptions of Google's patents and other intellectual property herein are intended to provide illustrative, non-exhaustive examples of some of the areas to which the patents and applications are currently believed to pertain, and is not intended for use in a legal proceeding to interpret or limit the scope or meaning of the patents or their claims, or indicate that a Google patent claim(s) is materially required to perform or implement any of the listed items.

Purpose

This document describes the issues and solutions to address multiple DRM system interoperability with a single DASH ([CENC](#)) content library.

Summary

An overview of Widevine Modular DRM (the use of Encrypted Media Extensions, DASH and a CDM) is available [here](#).

Encryption

- The Common Encryption ([CENC](#)) specification dictates content must be encrypted using a standard AES algorithm with a 128-bit Content Key, independent of any DRM system.
 - The key size is 128-bit.
 - Each key is associated with a 128-bit unique Key ID.
 - The Key ID field acts as a unique identifier for the Content Key. It is used to maintain mapping of Content Keys to encrypted content.
 - In a live streaming use case, Content Keys must change at predefined time intervals governed by key rotation parameters.
 - For a given time interval, the correct Content Key must be used for decryption.
 - To keep track of the correct Content Key to be used, the Key ID is used to maintain the index of Content Keys to encrypted content.
- Encrypted content is annotated with an identification header, called a [PSSH \(Protection System Specific Header\)](#). CENC content requires the creation of a PSSH (specific to a DRM vendor) which is used for playback.
 - The PSSH is used to identify a specific DRM system and the metadata needed to identify the content in a license request.
 - The Widevine identification string in a PSSH is `edef8ba9-79d6-4ace-a3c8-27dcd51d21ed`
 - A list of DRM system IDs is available [here](#).

Content (encryption) Keys can be obtained from:

- Widevine Cloud License Service
 - A Content Key may be requested by specifying a Content ID in a key request.
 - The license services manages the mapping of Content ID to Content key
 - Access to a Content Key is always referred to by the Content ID.
- Using third-party Content Keys with a Widevine License Service
 - The Content Key does not originate from the Widevine Cloud License Service or the Widevine License SDK.

Playback

- Content playback is managed and controlled by the player application which interfaces with the DRM client and DRM license service.
- The [Content Decryption Module \(CDM\)](#) is the specific DRM client for a platform.
 - It obtains the PSSH information to generate a request and receive a license to decrypt content.
 - The implementation and operation of a CDM varies by DRM system.

Requirements

There are two requirements to enable use of encrypted content from other DRM systems (i.e. content that does not originate from the Widevine DRM system used for playback):

- **Widevine PSSH**
The presence of a Widevine PSSH box is required for compatibility and use with the Widevine DRM system.
- **The Content Key**
Content that is encrypted outside of the Widevine ecosystem will use content key(s) that are not stored within the Widevine ecosystem.
When license requests are made for such content, these Content Key(s) must be provided for proper license generation. This is accomplished by using the `content_key_specs` method as described in the [Widevine Proxy Integration document](#).

Construction of the Widevine PSSH

The format of a Widevine PSSH box is as follows:

```
4 byte size (includes the header)
4 byte type (0x 70 73 73 68)
1 byte version (0x 00)
3 bytes flags (0x 00 00 00)
16 byte system ID (Widevine - edef8ba979d64acea3c827dcd51d21ed)
4 byte data size (lets call this N)
N bytes of serialized protection-system-specific data
```

Field	Byte size	Value	Description
size	4 byte	Calculated based on input variables	size of the entire PSSH box, includes the header

type	4 byte	0x 70 73 73 68	stands for "pssh"
version	1 byte	0x 00	stands for version 0
flags	3 byte	0x 00 00 00	stands for flags = 0
system ID	16 byte	edef8ba979d64acea3 c827dcd51d21ed	Widevine system ID
data size	4 byte	Calculated based on input variables	N bytes of PSSH data string
protection system specific data string	N bytes	Serialized WidevineCencHeader proto message	See section below

The format of the PSSH data is defined by the WidevineCencHeader Google Protocol Buffer as shown below.

For additional details, please refer to the [Widevine Common Encryption API](#) document.

```

message WidevineCencHeader {
  enum Algorithm {
    UNENCRYPTED = 0;
    AESCTR = 1;
  };
  optional Algorithm algorithm = 1;
  repeated bytes key_id = 2;

  // Content provider name.
  optional string provider = 3;

  // A content identifier, specified by content provider.
  optional bytes content_id = 4;

  // Track type. Acceptable values are SD, HD and AUDIO. Used to
  // differentiate content keys used by an asset.
  optional string track_type = 5;

  // The name of a registered policy to be used for this asset.
  optional string policy = 6;

  // Crypto period index, for media using key rotation.
  optional uint32 crypto_period_index = 7;
}

```

A [reference Python script](#) to demonstrate the construction of a Widevine PSSH is included in the Appendix of this document.

PSSH Insertion

This section describes methods to annotate content with a Widevine PSSH:

Insertion of PSSH into the content

The PSSH box can be constructed and inserted in the content header at the time of content encryption.

For example, the [Widevine eDASH-packager](#) generates the PSSH box by embedding the information used for encryption and inserts it in the content header.

Insertion of PSSH into the MPD

Modification of previously-encrypted content by third-party DRM systems is non-trivial and time intensive. In such cases, the PSSH box can be constructed independently ([see above](#)) and inserted in the content's DASH MPD file.

The MPD is a content playlist, from which the player application can be configured to retrieve the PSSH information.

Example of PSSH insertion in MPD

Note: The value of the Widevine PSSH must be base64-encoded.

```
<AdaptationSet mimeType="video/mp4" subsegmentAlignment="true">
  <ContentProtection schemeIdUri="http://example.com/myDrmSchema">
    <cenc:pssh>
      AAAAV3Bzc2gAAAAA7e+LqXnWSs6jyCfc1R0h7QAAADcIARIgMzI3OTcxNTIzOTZmNGY0Mj
      NlNWY1NDZlMmI1MjZlNW EaBWN3aXAxIgZ0ZXN0MDEqAkhE
    </cenc:pssh>
  </ContentProtection>

  <Representation id="142" codecs="avc1.4d4015" width="426" height="240"
  startWithSAP="1" bandwidth="282817">
    <BaseURL>oops_cenc-20121114-142.mp4</BaseURL>
    <SegmentBase indexRange="1739-2358" indexRangeExact="true">
      <Initialization range="0-1738"/>
    </SegmentBase>
  </Representation>
  <Representation>
    .....
  </Representation>
</AdaptationSet>
```

The Widevine [open-source DASH HTML5 player \(Shaka\)](#) supports the PSSH insertion in the MPD:

- [Sample MPD](#)
- Supported Classes
 - [shaka.player.DrmSchemeInfo](#)
 - [shaka.dash.mpd.ContentProtection](#)
- [Working Shaka Player code](#)

Playback using foreign content key(s)

The Widevine License service uses a 128-bit Content Key in order to generate a valid license.

- The Content Key is accompanied by a 128-bit Key ID.
- The Key ID is a unique identifier for a Key.

In the case of third-party encrypted content, this Content Key must be provided in the request to be used in the license output.

With the use of a self-generated Widevine PSSH and third-party Content Keys, it is important to ensure that the values in the generated PSSH must match the parameters of the license request:

- Key ID
- Content Key
- Track Type

Using the Widevine Cloud License Service

It is important to note:

- The content's PSSH parameters must match the license request parameters.
- The license request parameters cannot reference values that do not exist in the content's PSSH.

From the [Widevine Proxy Integration document](#), the table below denotes the available fields:

(Required fields are highlighted in yellow.)

Name	Value	Description
content_key_specs. track_type	string	A track type name. If content_key_specs is specified in the license request, make sure to specify all track types declared in the pssh explicitly.
content_key_specs. key	Base64 encoded string	Content key to use for this track. If specified, the track_type or key_id is required. This option allows the content provider to inject the content key for this track instead of letting Widevine license server generate or lookup a key.

content_key_specs. key_id	Base64 encoded string binary, 16 bytes	Unique identifier for the key.
content_key_specs. security_level	uint32	Defines client robustness requirements for playback. 1 - Software-based whitebox crypto is required. 2 - Software crypto and an obfuscated decoder is required. 3 - The key material and crypto operations must be performed within a hardware backed trusted execution environment. 4 - The crypto and decoding of content must be performed within a hardware backed trusted execution environment. 5 - The crypto, decoding and all handling of the media (compressed and uncompressed) must be handled within a hardware backed trusted execution environment. See Appendix .
content_key_specs. required_output_protection.hdcv	string - one of: HDCP_NONE, HDCP_V1, HDCP_V2	Indicates whether HDCP is required. See Appendix .

content_key_specs.track_type

This is a Widevine-specific parameter that may not be present in other DRM implementations. It represents the resolution of a track and can hold one of three values - HD, SD and AUDIO.

The presence of a track_type value allows for greater granularity on the application of license rules.

A missing track_type field will prevent restriction of playback for a specific track for the following parameters:

- HDCP
- EME security level

content_key_specs.key_id

This parameter represents the unique identifier for a Content Key and is required to maintain the sequence of Content Keys to be used (with key rotation).

The actual value of a Key ID is immaterial as long as it is associated correctly to the content key. The association of Key ID to third-party Content Keys is the responsibility of the content provider.

If key rotation is not required, the Key ID can be set to a static value (e.g. 0). We recommend the use of the Key ID for identification purposes.

- A combination of track_type and Key ID will accurately identify a media track (via its PSSH) within a compilation of media described in an MPD.

Example of key_id

The code below demonstrates the use of track_type (in 3 variations) in combination with Key ID in order to enforce device security level and output protection requirements.

```
"content_key_specs":[
  {
    "key_id": "MGRmMjVjYzUtOWFkMS01MjhiLTkwZGMtNTYyY2QxODExODNi",
    "track_type": "SD",
    "key": "Pv38THb6rTQbIpzg04/qag==",
    "security_level": "3",
    "required_output_protection.hdc": "HDCP_NONE"
  },
  {
    "key_id": "NzVlYThiYWMTZjg3Yy01YzMxLWFmYTtNWY3YmUwNjE1MGU5",
    "track_type": "HD",
    "key": "xiE14IkMIRKxSPBQL+HksA==",
    "security_level": "5",
    "required_output_protection.hdc": "HDCP_V1"
  },
  {
    "key_id": "NDczMwNhNzItOWExYy01YjQ1LTJhMTMtMGY4MDJiMDRiMWNj",
    "track_type": "AUDIO",
    "key": "y9DDm30ttT07GqXfxKMwdQ=="
  }
]
```

The Widevine ecosystem allows you to encrypt each individual track type with a different Content Key. Each Content Key is paired with a distinct Key ID.

The tabular version of the configuration in the above example is as follows:

Track type	Security Level	Output Protection
SD	3	HDCP_NONE
HD	5	HDCP_V1
AUDIO	No restriction	No restriction

Including track type and Key ID values when annotating third-party encrypted content with a Widevine PSSH is highly recommended for the application of business rules on a granular level.

Case Study - Using PlayReady-generated Key ID

Playready-generated SmoothStreaming content that is encrypted using [CENC](#) has a unique way of representing Key IDs. [This Wikipedia link](#) explains the Playready GUID structure.

Problem Description

The Widevine DRM ecosystem expects the Key ID to be in Big Endian format, whereas the Microsoft Key ID is in inverted byte order.

The Key ID used to encrypt content in a commonly encrypted environment can be extracted from the PSSH data typically present in the header segment of the encrypted content.

When the Key ID is Playready-generated, the binary/hex Key ID string extracted from the PSSH is found to be in a non-Big-Endian byte order format. If used as-is when making a license request with the Widevine License server, it would result in a playback failure.

There are two methods to address this issue:

Generate a Widevine PSSH using a converted GUID

In order to make Playready encrypted content compatible with the Widevine CDM, the content needs to be re-packaged with a Widevine PSSH. Re-packaging in this context does not necessarily mean re-encryption, [CENC](#) allows for content to contain multiple PSSH boxes, each representing a different DRM vendor.

This translates to

- building a PSSH box with the same Key ID as that of the Playready PSSH
- providing the same Key ID and Content Key values when making a license request

Content Re-packaging

In order to annotate pre-encrypted Playready content with a Widevine PSSH, it is necessary to ensure that the correct format of the Key ID is used.

In other words, the Key ID should be converted to Big Endian format when generating a Widevine PSSH. A sample script to convert Microsoft GUID to Big Endian format is provided in the [Appendix](#) below.

Content Playback

A base64-encoded version of the converted Key ID needs to be specified in the license request using [content_key_specs](#).

Using an alternate set of Key IDs

The only imperative is to ensure that the Key ID and content keys match in both the PSSH and license request.

You may use a separately-assigned range of Key IDs to be associated with the content key.

- This approach avoids the complexity of converting Key IDs from Microsoft GUID formats.
- The onus of maintaining the appropriate sequence of Key IDs, mapping of Key ID to content key is outside the scope of the Widevine ecosystem.

Successful decryption of content will take place as long as the Key ID in license issued matches the Key ID in the PSSH of the content.

Appendix

Generation of a Widevine PSSH Box

The code below is a reference Python script to construct a Widevine PSSH. This script requires the [WidevineCencHeader proto](#) to be built and included as a dependency.

```
#!/usr/bin/python
"""Reference Widevine Modular DRM PSSH box generator.

Reference script to generate a Widevine PSSH using custom content keys."""

import base64
import binascii
import struct
import widevine_pssh_pb2

WIDEVINE_SYSTEM_ID = "edef8ba979d64acea3c827dcd51d21ed"

### PSSH Generation for SD_HD tracks ###

widevine_pssh_string = widevine_pssh_pb2.WidevineCencHeader()
widevine_pssh_string.algorithm = 1
widevine_pssh_string.key_id.append(bytes("<your key id>"))
widevine_pssh_string.provider = "<your provider>"
widevine_pssh_string.content_id = bytes("<optional content id>")
widevine_pssh_string.track_type = "<SD, HD OR Audio>"

data = widevine_pssh_string.SerializeToString()
pssh_box_size = 12 + 16 + 4 + len(data)
pssh = struct.pack(">III", pssh_box_size, 0x70737368, 0)
pssh += binascii.unhexlify(WIDEVINE_SYSTEM_ID)
pssh += struct.pack(">I", len(data))
pssh += data

assert pssh_box_size == len(pssh)

pssh_hex = binascii.hexlify(pssh)
pssh_base64 = base64.standard_b64encode(pssh)

## Verification
print "\npssh proto : \n" + str(widevine_pssh_string)
print "Hex pssh data (to be used to package content): \n" + binascii.b2a_hex(data)
print "Hex pssh box (to be used as the Widevine PSSH header in prepackaged content): \n" +
pssh_hex
```

```
print "Base64 of pssh (to be used in the MPD):  \n" + pssh_base64
```

Output of the above script with sample values would look like:

```
pssh proto :
algorithm: AESCTR
key_id: "32797152396f4f423e5f546e2b526e5a"
provider: "widevine_test"
content_id: "test01"
track_type: "HD"

Hex pssh data (to be used to package content):
0801122033323739373135323339366634663432336535663534366532623532366535611a0d7769646576696e65
5f7465737422067465737430312a024844
Hex pssh box (to be used as Widevine PSSH header in prepackaged content):
0000005f7073736800000000edef8ba979d64acea3c827dcd51d21ed0000003f0801122033323739373135323339
366634663432336535663534366532623532366535611a0d7769646576696e655f7465737422067465737430312a
024844
Base64 of pssh (to be used in the MPD):
AAAAX3Bzc2gAAAAA7e+LqXnWSs6jyCfc1R0h7QAAAD8IARiGmZI3OTcxNTIzOTZmNGY0MjNlNWY1NDZlMmI1MjZlNW Ea
DXdpZGV2aW5lX3Rlc3QiBnRlc3QwMSoCSEQ=
```

Client security level (content_key_specs.security_level)

A license may be issued with a requirement for the minimum client security level.

The table below illustrates the general mapping between the EME security level definitions and Widevine device robustness levels.

Definition	Value	Widevine Device Security Level
SW_SECURE_CRYPTO	1	3
SW_SECURE_DECODE	2	3
HW_SECURE_CRYPTO	3	2
HW_SECURE_DECODE	4	1
HW_SECURE_ALL	5	1

HDCP Output Matrix

The table below is a results matrix for setting output protection HDCP_V1 in the license request.

OP Error indicates playback will fail with an output protection error.

HDCP v1 Required	No External Display	HDCP-compliant External Display	Non-HDCP-compliant External Display	Analog External Display	ChromeCast	AirPlay
ChromeOS ARM	OK	OK	OP Error	N/A	OP Error	N/A
ChromeOS x86	OK	OP Error	OP Error	N/A	OP Error	N/A
Linux	OP Error	OP Error	OP Error	OP Error	OK	N/A
Mac	OK	OP Error	OP Error	OP Error	OK	OP Error
Windows	OK	OK	OP Error	OP Error	OK	N/A

Conversion to Widevine-compatible Key ID format

```
#!/usr/bin/python

import binascii
import uuid

def byteswap_hex_guid(hex_guid):
    """Converts a little-endian hexadecimal GUID to big-endian, or vice-versa."""
    return uuid.UUID(bytes_le=binascii.unhexlify(hex_guid)).hex

hex_guid = '01234567890123456789012345678901' # YOUR GUID GOES HERE
assert len(hex_guid) == 32

print 'original =', hex_guid
print 'swapped  =', byteswap_hex_guid(hex_guid)
```