Widevine DRM
Common Encryption API

Version 1.6

Revision History

Date	Description
10/08/2014	● Updated test player and content urls.
07/30/2013	● Updated Key Id (tracks.key_id). Type is 128 bit binary returned as Base64 encoded string. ● Updated Content Id. Type is binary specified as a Base64 encoded string. Size cannot exceed 1024 bytes.
08/02/2013	● Change single Widevine PSSH to an array containing Widevine and PlayReady. New field in response is called content_protection.
03/21/2014	● Add new fields to support live streaming with key rotation. ○ first_crypto_period_index ○ crypto_period_count
01/13/2015	● Added section describing inserting a PSSH into a MPD ● Updated demo player section with Widevine Shaka Player links
04/21/2015	● Updated signature requirements
01/22/2016	● Added drm_types array to the encryption request to support PlayReady.
02/29/2016	● Add crypto_period_seconds.
06/28/2016	● Updated Widevine proto.
03/05/2017	● Updated to add optional group_id in support of license embedding.
03/29/2018	● Updated with changes to the pssh in support of entitlement keys. ● Added appendix describing entitlement keys.
08/20/2018 v1.6	● Details on entitlement licensing. ● Updated WidevinePsshData proto to include deprecated fields.

Contents

Introduction

This document specifies an interface for adding support for Google's Widevine encryption to a video processing system. The interface allows the encryption process to request and receive the Widevine header and encryption key.

Description

The encryption process sends an HTTP(s) request to the Key server. The request contains the unique content identifier (ContentId). The ContentId is owned by the encryption process. The Key server will respond with Protection System Specific Header (PSSH) data for Widevine and Content Key(s) that are associated with the ContentId.

- All Base64 encoded strings use the standard encoding specified in RFC 4648.
- Widevine SystemID for the PSSH
  - `edef8ba979d64acea3c827dcd51d21ed`
- DASH MPD Content Protection type: `"widevine"`

Details

Authentication

The protocol described in this documents supports the ability for the client and server to optionally sign the message. The method described here is the client will sign the request by using the client's AES key and Widevine will sign the response using Widevine's RSA key.

Request

The request is an HTTP POST request to the KeyServer URL.
The posted data is a JSON formatted (optionally signed) request with the following syntax:

```
{
  "request":"<message>",
  "signature":"<signature>",
  "signer":"<signer name>"
}
```

Name	Value	Description
request	Base64 encoded string.	Required field. The actual message. This is a serialized JSON message containing the request or the response. The message is in the clear.

---

signature	Base64 encoded string. RFC 4648	Optional field. Required for access to Widevine Cloud License Service. AES CBC encryption of the SHA1 hash of <message>.
signer	string	Required. Identifies the entity sending the message.

The clear <request> is a JSON formatted string containing the content id and the track types. Content may have one or more track types, for example, the audio track may be encrypted with a different key than the SD video track. Or the content may have an SD video track and an HD video track encrypted with different keys.

```
{
  "content_id":"dGVzdC1jb250ZW50MQ==",
  "tracks":[
    {
      "type":"HD"
    },
    {
      "type":"SD"
    },
    {
      "type":"Audio"
    }
  ],
  "drm_types":[
    "WIDEVINE"
  ]
}
```

Name	*Value*	*Description*
type	One of: HD, SD, AUDIO (may not be limited to this set of types)	Required. Label to identify the track type. It's OK for multiple requests to specify the same track type. In this case, the same key id and key is returned for each track of the same type.
policy	string - Ascii, max size = 1024 bytes.	<DEPRECATED> Name of a previously stored policy to use for this content. The policy contains playback duration information and output protection information for SD, HD and AUDIO. If <policy> is not specified,

		the content provider must provide all policy attributes at the time of license acquisition.
content_id	Base64 encoded string - Binary, max size = 1024 bytes.	Required. Unique identifier of the content. The content_id will be available in the license request.
token	Optional base64 encoded string.	If present, the 72-byte token from a Widevine keybox. Content keys in the response will be AES- or 3DES-encrypted using the device key associated with this token, and thus the response will only usable by a device with that keybox.
rsa_public_key	Optional RSA public key in Base64 DER-encoded PKCS#1 format.	If present, a random session_key encrypted with this public key will be included in the response. Content keys in the response will be AES-ECB encrypted with the session key.
first_crypto_period_index	32 bit unsigned int	Used for key rotation only. The time value for the first key period for which keys are requested, divided by the period duration. Perhaps epoch or media time.
crypto_period_count	32 bit unsigned int	Used for key rotation only. Number of keys requested.
crypto_period_seconds	32 bit unsigned int	Used for key rotation only. Number of seconds in each crypto period.
drm_types	Optional enumerated drm type identifier.	Used to request pssh data conforming to non-widevine drm.
group_id	An optional list of group name strings.	Used to create content groups for packaging.. When provided in a request, any returned Widevine PSSH will contain grouping information. The format of group_id is defined by the provider. Example: "ChannelGroup2000"

Signing the request

- Generate the JSON-string.
  Json::StyledWriter writer;
  std::string clear_message = writer.write(root);

- Sign the message.

- ○ Generate SHA1 hash of the message. Result is 20 bytes string.
- ○ Encrypt the 20 byte hash.
  - ■ For AES: CBC, PKCS5 padding
  - ■ For RSA: RSA-PSS, PKCS1.5 padding, Salt length=20.

Multiple requests using the same Content Id with different track types.

It is possible for the client to send multiple requests for content keys with the same content Id (c_id)) but with different track types. For example,

- request #1: content Id = CID, track type = SD.
- request #2: content Id = CID, track type = AUDIO.

In this case, the KeyServer will return a response for each request as follows:

- response #1: status = OK, content Id = CID, key Id = KID_1, content key = CKey_1, track type = SD
- response #2: status = OK, content Id = CID, key Id = KID_2, content key = CKey_2, track type = AUDIO.

Multiple requests using the same Content Id with same track types.

If multiple requests are sent with the same content Id and same track type, the first response will contain the same key Id and content key. The second response will also succeed and field indicating already used.

- request #1: content Id = CID, track type = SD.
- request #2: content Id = CID, track type = SD.

In this case, the KeyServer will return a response for each request as follows:

- response #1: status = OK, content Id = CID, key Id = KID_1, content key = CKey_1, track type = SD.
- response #2: status = OK, content Id = CID, key Id = KID_1, content key = CKey_1, track type = SD, already_used = true.

Response

If signed, the response is a JSON formatted string.

```
{
  "response":"<message>",
  "signature":"<signature>"
}
```

Where <message> is a JSON formatted string. For each track type in the request, there is a key and key_id in the response. The actual <message> in the response is serialized JSON:

```
{
  "status":"OK",
  "drm":[
    {
      "type":"WIDEVINE",
      "system_id":"edef8ba9-79d6-4ace-a3c8-27dcd51d21ed"
    },
    {
      "type":"PLAYREADY",
      "system_id":"79f0049a-4098-8642-ab92-e65be0885f95"
    }
  ],
  "tracks":[
    {
      "type":"HD",
      "key_id":"a2V5aWQtMDAx",
      "key":"MTIzNDU2Nzg5MDEyMzQ1Ng==",
      "iv":"MDAwMDAwMDAwMDAwMA==",
      "pssh":[
        {
          "drm_type":"WIDEVINE",
          "data":"c2FtcGxlLXBzc2g="
        },
        {
          "drm_type":"PLAYREADY",
          "data":"c2FtcGxlLXBzc2g="
        }
      ]
    },
    {
      "type":"SD",
      "key_id":"a2V5aWQtMDAy",
      "key":"MjM0NTY3ODkwMTIzNDU2MQ==",
      "iv":"MDAwMDAwMDAwMDAwMA==",
      "pssh":[
        {
          "drm_type":"WIDEVINE",
          "data":"c2FtcGxlLXBzc2g="
```

```
        },
        {
          "drm_type":"PLAYREADY",
          "data":"c2FtcGxlLXBzc2g="
        }
      ]
    },
    {
      "type":"Audio",
      "key_id":"a2V5aWQtMDAz",
      "key":"MzQ1Njc4OTAxMjM0NTYxMg==",
   "iv":"MDAwMDAwMDAwMDAwMDAwMA==",
      "pssh":[
        {
          "drm_type":"WIDEVINE",
          "data":"c2FtcGxlLXBzc2g="
        },
        {
          "drm_type":"PLAYREADY",
          "data":"c2FtcGxlLXBzc2g="
        }
      ]
    }
  ],
  "content_id":"dGVzdC1jb250ZW50MQ=="
}
```

Name	Value	Description
status	string See Status Codes table below.	If successful, OK is returned. Otherwise an error code is returned to indicate failure.
session_key	Base64 encoded binary string	Optional AES-128 bit random key used to encrypt the content keys. The session_key is encrypted with the client's RSA key specified in the request.
drm	array of JSON struct.	A list of DRM systems.
drm.type	String. one of: "WIDEVINE" "PLAYREADY"	The DRM system.
drm.system_id	string	Registered system Id for the DRM system.
tracks.pssh	array of JSON struct	A list of PSSH, one per DRM system.

tracks.pssh.drm_type	string	The DRM system for a specific piece of PSSH
tracks.pssh.data	Base64 encoded string	PSSH (protection system specific header) data. The PSSH data is unique per track.
tracks.key_id	Base64 encoded string Binary, 16 bytes	Unique identifier for the key.
tracks.type	One of: HD SD AUDIO	Identifies the track type encrypted with this key. These values are extracted from the request.
tracks.key	Base64 encoded string Binary, 16 bytes	Content key. If the session_key is specified, the content key is AES-ECB encrypted with the session key. The session key is generated by the DRM server and encrypted with the content provider's public key using RSA-OAEP(sha1, mgf1sha1). The encrypted key is Base64 encoded.
already_used	Boolean	Set to true if the key_id was previously issued. This can happen if a previous key request from the same signer with the same content_id and track_type was sent.

Signing the response
- Generate the JSON-string.
    Json::StyledWriter writer;
    std::string clear_message = writer.write(root);

- Sign the message.
    RSA-PSS

Status codes

Code	Description
OK	Success.
SIGNATURE_FAILED	The server was unable to authenticate the message. Perhaps: ● The specified <signer> is unknown ● The <signature> was not specified, but is required in this case.

CONTENT_ID_MISSING	<content_id> field is missing in the request.
POLICY_UNKNOWN	The specified <policy> was not found for this <signer>.
TRACK_TYPE_MISSING	<tracks.type> is missing in the request.
TRACK_TYPE_UNKNOWN	The specified <tracks.type> is not one of the allowed values.
MALFORMED_REQUEST	The request is not formatted correctly.
ACCESS_DENIED	The content provider is not permitted to perform the operation (for example, an operation like "retrieve content keys").

PSSH

Below is the Widevine PSSH format for content providers who wish to synthesis the PSSH rather than using the ones returned by the API. The structure below is a protocol buffer (see https://developers.google.com/protocol-buffers/). The process is:

1) Build the protocol buffer message below.
2) Serialize the message to bytes.
3) Base64 encode the bytes.

```
syntax = "proto2";

package video_widevine;
option java_package = "com.google.video.widevine.protos";

message WidevinePsshData {
  enum Type {
    SINGLE = 0;         // Single PSSH to be used to retrieve content keys.
    ENTITLEMENT = 1;    // Primary PSSH used to retrieve entitlement keys.
    ENTITLED_KEY = 2;   // Secondary PSSH containing entitled key(s).
  }

  message EntitledKey {
    // ID of entitlement key used for wrapping |key|.
    optional bytes entitlement_key_id = 1;
    // ID of the entitled key.
    optional bytes key_id = 2;
    // Wrapped key. Required.
    optional bytes key = 3;
    // IV used for wrapping |key|. Required.
    optional bytes iv = 4;
  }

  // Entitlement or content key IDs. Can only be present in SINGLE or
  // ENTITLEMENT PSSHs. May be repeated to facilitate delivery of
  // multiple keys in a single license. Cannot be used in conjunction
  // with content_id or group_ids, which are the preferred mechanism.
  repeated bytes key_ids = 2;

  // Content identifier which may map to multiple entitlement or content key
  // IDs to facilitate the delivery of multiple keys in a single license.
  // Cannot be present in conjunction with key_ids, but if used must be
  // in all PSSHs.
  optional bytes content_id = 4;

  // Crypto period index, for media using key rotation. Always corresponds
  // to the content key period. This means that if using entitlement
  // licensing the ENTITLED_KEY PSSHs will have sequential
  //  crypto_period_index's, whereas the ENTITLEMENT PSSHs will have gaps
  // in the sequence. Required if doing key rotation.
  optional uint32 crypto_period_index = 7;
```

```
// Protection scheme identifying the encryption algorithm. The protection
// scheme is represented as a uint32 value. The uint32 contains 4
// bytes each representing a single ascii character in one of the 4CC
// protection scheme values. To be deprecated in favor of signaling
// from content.
// 'cenc' (AES-CTR) protection_scheme = 0x63656E63,
// 'cbc1' (AES-CBC) protection_scheme = 0x63626331,
// 'cens' (AES-CTR pattern encryption) protection_scheme = 0x63656E73,
// 'cbcs' (AES-CBC pattern encryption) protection_scheme = 0x63626373.
optional uint32 protection_scheme = 9;

// Optional. For media using key rotation, this represents the duration
// of each crypto period in seconds.
optional uint32 crypto_period_seconds = 10;

// Type of PSSH. Required if not SINGLE.
optional Type type = 11 [default = SINGLE];

// Key sequence for Widevine-managed keys. Optional.
optional uint32 key_sequence = 12;

// Group identifiers for all groups to which the content belongs. This can
// be used to deliver licenses to unlock multiple titles / channels.
// Optional, and may only be present in ENTITLEMENT and ENTITLED_KEY
// PSSHs, and not in conjunction with key_ids.
repeated bytes group_ids = 13;

// Copy/copies of the content key used to decrypt the media stream in
// which the PSSH box is embedded, each wrapped with a different
// entitlement key. May also contain sub-licenses to support devices
// with OEMCrypto 13 or older. May be repeated if using group
// entitlement keys. Present only in PSSHs of type ENTITLED_KEY.
repeated EntitledKey entitled_keys = 14;

/////////////////////////// Deprecated Fields ///////////////////////////
enum Algorithm {
  UNENCRYPTED = 0;
  AESCTR = 1;
};
optional Algorithm algorithm = 1 [deprecated = true];
optional string provider = 3;
optional string track_type = 5 [deprecated = true];
optional string policy = 6 [deprecated = true];
optional bytes grouped_license = 8 [deprecated = true];
}
```

PSSH in the MPD

In compliance with [DASH IF Interoperability Points](), Widevine supports insertion of the PSSH in the DASH MPD file.

The PSSH box, including its header, must be base64 encoded and placed in the <cenc:pssh> element nested under the <ContentProtection> element. See the example below:

```
<AdaptationSet mimeType="audio/mp4">
    <ContentProtection
schemeIdUri="urn:uuid:edef8ba9-79d6-4ace-a3c8-27dcd51d21ed">
        <cenc:pssh>AAAANHBzc2gAAAAA7e+</cenc:pssh>
    </ContentProtection>
```

Test Player

To help integration and verification, the following can be used to playback Widevine CENC content:

Demo URLs for players
http://shaka-player-demo.appspot.com/

Content URL
http://storage.googleapis.com/wvmedia/cenc/tears.mpd

Widevine has published an open-source HTML5 DASH player, available at:
github site: https://github.com/google/shaka-player
changelog: https://github.com/google/shaka-player/blob/master/CHANGELOG.md
hosted demo: http://shaka-player-demo.appspot.com/
hosted copy of documentation: http://shaka-player-demo.appspot.com/docs/index.html
hosted copy of the final tutorial from the docs: http://turtle-tube.appspot.com/

Appendix

Sample Request
Provider: widevine_test
Content ID: fkj3ljaSdfalkr3j
Policy: empty string

Here is the command to obtain keys for this sample asset:

```
wget -O /tmp/key --post-data '{"request":
"ewogICJjb250ZW50X2lkIjogIlptdHNNnhxWVZZOa1ptRnNhM0l6YWc9PSIsCiAgInRyYWNrcyI6I
FsKICAgIHsgInR5cGUiOiAiU0QiIH0sCiAgICB7ICJ0eXBlIjogIkhEIiB9LAogICAgeyAidHlwZSI
6ICJBVURJTyIgfQogIF0sCiAgImRybV90eXBlcyI6IFsgIldJREVWSU5FIiBdLAogICJwb2xpY3kiO
iAiIgp9Cg==", "signature": "kwVLL4xVh9mnlZlPqiEWN0E+FsvG0y+/oy451XXeIMo=",
"signer": "widevine_test" }'
http://license.uat.widevine.com/cenc/getcontentkey/widevine_test
```

The expected response should be:

```
{"response":"eyJzdGF0dXMiOiJPSyIsImRybSI6W3sidHlwZSI6IldJREVWSU5FIiwic3lzdGVtX
2lkIjoiZWRlZjhiYTk3OWQ2NGFjZWEzYzgyN2RjZDUxZDIxZWQifV0sInRyYWNrcyI6W3sidHlwZSI
6IlNEIiwia2V5X2lkIjoiQXBTNVdaMTFYZUs3OFAzS0P6WHF0dz09Iiwia2V5IjoiTzlvdlFEUk1mZ
TloUWllNXdQQStKZz09IiwicHNzaCI6W3siZHJtX3R5cGUiOiJXSURFVklORSIsImRhdGEiOiJJaEJ
tYTJvemJHcGhVMlJtWVd4cmNqTnFTT1BjbFpzRyJ9XSwiZW50aXRsZWRfa2V5IjpbXX0seyJ0eXBlI
joiSEQiLCJrZXlfaWQiOiI2MmRxdThzMFhwYTd6MkZtTVBHajJnPT0iLCJrZXkiOiJFQXRzSUpPZGQ
1cEZpUlVyVjlMYXl3PT0iLCJwc3NoIjpbeyJkcm1fdHlwZSI6IldJREVWSU5FIiwiZGF0YSI6IkloQ
m1hMm96YkdwaFUyUm1ZV3hyY2pOcVNPUGNsWnNHIn1dLCJlbnRpdGxlZF9rZXkiOltdfSx7InR5cGU
iOiJBVURJTyIsImtleV9pZCI6Ilk1Mm9EUEk3VmZPNHlyUDJUUHBkOWc9PSIsImtleSI6IklwOWZLY
lpENGdNQVN6REU2dk5JOUE9PSIsInBzc2giOlt7ImRybV90eXBlIjoiV0lERVZJTkUiLCJkYXRhIjo
iSWhCbWEyb3piR3BoVTJSbVlXeHJjak5xU09QY2xac0cifV0sImVudGl0bGVkX2tleSI6W119XSwiY
WxyZWFkeV91c2VkIjp0cnVlfQ=="}
```

Decode the response
```
cut -d: -f2 /tmp/key | base64 -d -i
```

```
{"status":"OK","drm":[{"type":"WIDEVINE","system_id":"edef8ba979d64acea3c827dc
d51d21ed"}],"tracks":[{"type":"SD","key_id":"ApS5WZ11XeK78P3KP6Xqtw==","key":"
O9ovQDRMfe9hQie5wPA+Jg==","pssh":[{"drm_type":"WIDEVINE","data":"IhBma2ozbGphU
2RmYWxrcjNqSOPclZsG"}],"entitled_key":[]},{"type":"HD","key_id":"62dqu8s0Xpa7z
2FmMPGj2g==","key":"EAtsIJQPd5pFiRUrV9Layw==","pssh":[{"drm_type":"WIDEVINE","
data":"IhBma2ozbGphU2RmYWxrcjNqSOPclZsG"}],"entitled_key":[]},{"type":"AUDIO",
"key_id":"Y52oDPI7VfO4yrP2TPpd9g==","key":"Ip9fKbZD4gMASzDE6vNI9A==","pssh":[{
"drm_type":"WIDEVINE","data":"IhBma2ozbGphU2RmYWxrcjNqSOPclZsG"}],"entitled_ke
y":[]}],"already_used":true}
```

Signing

The signature is generated as follows.

1. Take the raw value of the "request" field (not in base64 encoding).

   {"policy": "test_policy", "tracks": [{"type": "SD"}, {"type": "HD"}, {"type": "AUDIO"}], "content_id": "dGVzdF9pZF9GOUZQQzBOTw=="}

2. Compute its SHA1 hash (this example uses widevine_test keys).

   306814a9_12abd779_de5dd373_89ca45d9_7328e89c

3. Encrypt the hash with AES CBC. The result is the signature:

   46f1ee6e3660acb081106c78e45e190464d87e5f8f7c79143c34da0308ec4bfc

4. Base64 encode the signature:

   RvHubjZgrLCBEGx45F4ZBGTYfl+PfHkUPDTaAwjsS/w=

5. Add signature to the request:

   *{"request":
   "eyJwb2xpY3kiOiAidGVzdF9wb2xpY3kiLCAidHJhY2tzIjogW3sidHlwZSI6ICJTRCJ9LCB
   7InR5cGUiOiAiSEQifSwgeyJ0eXBlIjogIkFVRElPIn1dLCAiY29udGVudF9pZCI6ICJkR1Z
   6ZEY5cFpGOUdPUdPVVpRUXpCT1R3PT0ifQ==",
   "signer": "widevine_test",
   "signature": "RvHubjZgrLCBEGx45F4ZBGTYfl+PfHkUPDTaAwjsS/w="}*

For integration tests, you can use "widevine_test" as the signer. Its AES key is

```
0x1a, 0xe8, 0xcc, 0xd0, 0xe7, 0x98, 0x5c, 0xc0,
0xb6, 0x20, 0x3a, 0x55, 0x85, 0x5a, 0x10, 0x34,
0xaf, 0xc2, 0x52, 0x98, 0x0e, 0x97, 0x0c, 0xa9,
0x0e, 0x52, 0x02, 0x68, 0x9f, 0x94, 0x7a, 0xb9
```

The IV is

```
0xd5, 0x8c, 0xe9, 0x54, 0x20, 0x3b, 0x7c, 0x9a,
0x9a, 0x9d, 0x46, 0x7f, 0x59, 0x83, 0x92, 0x49
```

Entitlement Keys

Entitlement Licensing is a feature designed to improve the handling of content that requires key rotation and content grouping. More information will be made available in the Widevine Group Licensing document (coming soon).