



Common Encryption API for Widevine DRM

Version 1.5

History

Date	Changes
10/08/2014	<ul style="list-style-type: none">• Updated test player and content urls.
07/30/2013	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• Change single Widevine PSSH to an array containing Widevine and PlayReady. New field in response is called content_protection.
03/21/2014	<ul style="list-style-type: none">• Add new fields to support live streaming with key rotation.<ul style="list-style-type: none">◦ first_crypto_period_index◦ crypto_period_count
01/13/2015	<ul style="list-style-type: none">• Added section describing inserting a PSSH into a MPD• Updated demo player section with Widevine Shaka Player links
04/21/2015	<ul style="list-style-type: none">• Updated signature requirements
01/22/2016	<ul style="list-style-type: none">• Added drm_types array to the encryption request to support PlayReady.
02/29/2016	<ul style="list-style-type: none">• Add crypto_period_seconds.
06/28/2016	<ul style="list-style-type: none">• Updated Widevine proto.

© 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.

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>
}
```

<i>Name</i>	<i>Value</i>	<i>Description</i>
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 <message> is a JSON formatted string containing the content id and the track types. A content may have one or more track types, for example, the audio track may be encrypted with a different key than the video track. Or the content may have an SD video track and an HD video track encrypted with different keys.

```
{
  "content_id": "vid0000001",
  "policy": "test_policy",
  "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 -	Optional.

	Ascii, max size = 1024 bytes.	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.

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. .

```
{
  "status": "OK",
  "content_id": "vid0000001",
  "session_key": "MTIzNDU2Nzg5MDEyMzQ1",
  "drm": [
    {
      "type": "widevine",
      "system_id": "edef8ba979d64acea3c827dcd51d21ed"
    },
  ],
}
```

```

{
  "type": "playready",
  "system_id": "79f0049a-4098-8642-ab92-e65be0885f95"
}
],
"tracks": [
  {
    "pssh":[
      {
        "drm_type": "widevine",
        "pssh_data": "aDM0aDM0MjNrMDAyODVubm5uZ2dnZzMyNDM0MmRmYWRm"
      },
      {
        "drm_type": "playready",
        "pssh_data": "TODO: Add XML data"
      }
    ]
    "key_id": "MTIzNDU2Nzg5MDExMTEyMQ==",
    "type": "HD",
    "key": "MTIzNDU2Nzg5MDAwMDAwMQ=="
  },
  {
    "pssh":[
      {
        "drm_type": "widevine",
        "pssh_data": "aDM0aDM0MjNrMDAyODVtbW1tZ2dnZzMyNDM0MmRmYWRm"
      },
      {
        "drm_type": "playready",
        "pssh_data": "TODO: Add XML data"
      }
    ]
    "key_id": "MTIzNDU2Nzg5MDIyMjlyMg==",
    "type": "SD",
    "key": "MTIzNDU4Nzg5MDAwMDAwMg==",
    "iv": "MTIzNDU2Nzg5MDEyMzQ1"
  },
  {
    "pssh":[
      {
        "drm_type": "widevine",
        "pssh_data": "aDM0aDM0MjNrMDAyODVtbW1tZ2dnZzMyNDM0MmRmYWRm"
      },
      {
        "drm_type": "playready",
        "pssh_data": "TODO: Add XML data"
      }
    ]
  }
]

```

```

    "key_id": "MTIzNDU2Nzg5MDMzMzMzMw==",
    "type": "AUDIO",
    "key": "MTIzNDU6Nzg5MDAwMDAwMg=="
  }
}
}

```

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: <ul style="list-style-type: none"> • 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	

PSSH

Below is the Widevine PSSH format for content providers who wish to synthesize 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.

```
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_deprecated = 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;

  // Optional protected context for group content. The grouped_license is a
  // serialized SignedMessage.
  optional bytes grouped_license = 8;

  // Protection scheme identifying the encryption algorithm.
  // Represented as one of the following 4CC values:
  // 'cenc' (AES-CTR), 'cbcl' (AES-CBC),
  // 'cens' (AES-CTR subsample), 'cbcs' (AES-CBC subsample).
  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;
}
```

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>AAAAANHBzc2gAAAAA7e+</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/>

Sample Request

Provider: widevine test

Content ID: fki3ljaSdfalkr3j

Policy: empty string

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

```
wget -O /tmp/key --post-data '{"request":
"ewogICJjb250ZW50X2lkIjogIlptdHFNbnhXWVZOa1ptRnNhM016YWc9PSIsCiAgInRyYWNrcyI6I
FsKICAgIHsgInR5cGUiOiAiU0QiIH0sCiAgICB7ICJ0eXB1IjogIkhhETIiB9LAogICAgYyAidHlwZSI6
6ICJBVBURJTYgIgogIF0sCiAgImRybV90eXB1cyI6IFsgIldJREVWVSU5FIiBdLAogICJwb2xpY3kiO
iAiIgp9Cg==", "signature": "kwVLL4xVh9mnlZlPqiEWN0E+FsvG0y+/oy45lXXeIMo=",
"signer": "widevine_test" }'
http://license.uat.widevine.com/cenc/getcontentkey/widevine test
```

The expected response should be:

```
{ "response": "eyJzdGF0dXMiOiJPSyIsImRybSI6W3sidHlwZSI6IldJREVWSU5FIiwic3lzdGVtX2lkIjoIYWRlZjhiYTk3OWQ2NGFjZWZlZyZyZyN2RjZDUxZDlXZWQifV0sInRyYWNrcyI6W3sidHlwZSI6IldlNEIiwia2V5X2lkIjoIQXBtbnVdaMTFYZUs3OFAzS1A2WHF0dz09Iiwia2V5IjoITzlvdlFEUklmZTloUWllNXdQQStKZz09IiwicHNzaCI6W3sia2ZlZDlXZWQifV0sImRhdGEiOiJlJDUyVTRUFlVXVWbWRKvJnpdS9EOXlqK2w2cmNhRfhkcFpHvJjJhVzVsWDNSbGMzUWlFRlpyYwPoc2FtRlRaRlpoYkd0eU0yb3FBbE5FTWdBPSJ9XX0seyJ0eXB1IjoISeQIJCjRzXlfaWQIoiI2MmRxdThzMfhwYtD6MkZtTVBHajJnPT0iLCjRzXkiOiJFQXRzSUprUGQ1cEzPulVyVjlMYXl3PT0iLCJwc3NoIjpbeyJkcmlfdHlwZSI6IldJREVWSU5FIiwizGF0YSI6IkNBRVNFT3RuYXJ2TE5GNld1ODloWmpEeG85b2FEWGRwWkdWMmFXNWxYM1JsYzNRaUVHWNJhak5zYw1GVFpHWMhiR3R5TTJvcUFraEVNz0E9In1dfSx7InR5cGUiOiJBVURJTyIsImtleV9pZCI6Iklk1Mm9EUEk3VmZPNHlyUDJUHHBkOWc9PSIsImtleSI6IklwOWZlZlplENGdNQVN6REU2dk5JOUe9PSIsInBzc2giOi0lt7ImRybV90eXB1IjoIv0IERVZJTkUiLCJkYXRhIjoIQ0FFU0VHT2R2XQXp5TzFYenVNCxo5a3o2WZZZYURYZHBaRlYyYVc1bFgzUmxjM1FpRUdadmFqTnNhBUZUWkdaaGJHdHlNMm9xQ1VGv1JfbbFBNz0E9In1dfV0sImFscmVhZlH1fdXNlZCI6dHJ1ZX0="}
```

Decode the response

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

```
{
  "status": "OK",
  "drm": [
    {
      "type": "WIDEVINE",
      "system_id": "edef8ba979d64acea3c827dc51d21ed"
    }
  ],
  "tracks": [
    {
      "type": "SD",
      "key_id": "ApS5WZ11XeK78P3KP6Xqtw==",
      "key": "O9ovQDRMfe9hQie5wPA+Jg==",
      "pssh": [
        {
          "drm_type": "WIDEVINE",
          "data": "CAESEAKUuVmddV3iu/D9yj+l6rcaDXdpZGV2aW5lX3Rlc3QiEGZrajNsamFTZGZhbGtyM2oqAlNEMGa="
        }
      ],
      "type": "HD",
      "key_id": "62dqu8s0Xpa7z2FmMPGj2g==",
      "key": "EAtsIJQPd5pFiRUrV9Layw==",
      "pssh": [
        {
          "drm_type": "WIDEVINE",
          "data": "CAESEOt narvLNF6Wu89hZjDxo9oaDXdpZGV2aW5lX3Rlc3QiEGZrajNsamFTZGZhbGtyM2oqAkHEMGa="
        }
      ],
      "type": "AUDIO",
      "key_id": "Y52oDPI7VfO4yrP2TPpd9g==",
      "key": "Ip9fKbZD4qMASzDE6vNI9A==",
      "pssh": [
        {
          "drm_type": "WIDEVINE",
          "data": "CAESEKtNarvLNF6Wu89hZjDxo9oaDXdpZGV2aW5lX3Rlc3QiEGZrajNsamFTZGZhbGtyM2oqAkHEMGa="
        }
      ]
    }
  ]
}
```

```
NE", "data": "CAESEGOdqAzyO1XzuMqz9kz6XfYaDXdpZGV2aW5lX3Rlc3QiEGZrajNsamFTZGZhbG  
tyM2oqBUFVRElPMgA="}}]], "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  
  7InR5cGUiOiAiSEQiSwgeyJ0eXB1IjogIkFVRElPIn1dLCAiY29udGVudF9pZCI6ICJkR1Z  
  6ZEY5cFpGOUdPVVpRUXpCT1R3PT0ifQ==",  
  "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
```