# Chrome EME Changes and Best Practices

January 2017

# Summary

Google Chrome is making changes to behavior in the Encrypted Media Extension (EME) implementation in Chrome browser to be more compliant with the spec. These changes are scheduled for **Chrome 58 (release date in Apr 2017)**.

This document discusses details of these changes and steps required for any content provider or HTML5 player developer/manager to stay current and avoid any service interruption.

Additionally, some best practices are outlined for to ensure web video application compatibility across different browsers and platforms.

# Upcoming Changes

## Codecs

Codecs will be mandatory in Chrome 58 and above.  Currently the Chrome browser does not care if this value is not set.
https://groups.google.com/a/chromium.org/forum/#!topic/blink-dev/aG_QGiPErlE

### What does the spec say?

(http://w3c.github.io/encrypted-media/#idl-def-mediakeysystemmediacapability)

> *The codecs parameter, which is required by most MIME types in order to determine the code, is usually necessary to specify exactly how the resource is encoded.*

### What will Chrome do starting from Chrome 58?

**Chrome will consider MIME types without codecs specified as unsupported**.  They will be ignored when considering if Widevine supports the specified configuration or not. If none of the remaining MIME types is supported, playback will fail.

### What should applications do?

Please specify the exact codec used by the encrypted content! That way if Chrome (or a particular version of Chrome) doesn't support it, the application will know before it starts to play and can potentially select an alternative.

### Examples

The following is currently supported in Chrome, but will fail starting in Chrome 58.

```
var widevineOptions = [
  {
    initDataTypes: [ 'cenc' ],
```

```
    videoCapabilities: [ {
      contentType: 'video/mp4'  // No codec specified.
    } ]
  }
];
navigator.requestMediaKeySystemAccess(
    'com.widevine.alpha', widevineOptions)
  .then(function(keySystemAccess) { … } );
```

The following is currently supported and will continue to be supported in Chrome 58.

```
var widevineOptions = [
  {
    initDataTypes: [ 'cenc' ],
    videoCapabilities: [ {
      contentType: 'video/mp4;codecs="avc1.4d401e"'
    } ]
  }
];
navigator.requestMediaKeySystemAccess(
    'com.widevine.alpha', widevineOptions)
  .then(function(keySystemAccess) { … } );
```

## Audio and Video Capabilities

In addition to the requiring specific codecs, also either **audioCapabilities** or **videoCapabilities** must be specified as part of a configuration when selecting a MediaKeySystem.
https://groups.google.com/a/chromium.org/forum/#!topic/blink-dev/aG_QGiPErlE

### What does the spec say?

(http://w3c.github.io/encrypted-media/#get-supported-configuration-and-consent, step 15)

> *If the videoCapabilities and audioCapabilities members in candidate configuration are both empty, return NotSupported.*

### What will Chrome do starting from Chrome 58?

Configurations that do not contain at **least one** videoCapabilities or audioCapabilities will be considered as **not supported.** If none of the configurations passed to requestMediaKeySystemAccess() contain at least one capability, the call will fail.

### What should applications do?

Always specify the required audioCapabilities and/or videoCapabilities in order to verify that the desired capabilities are indeed supported by the CDM.

### Examples

The following is currently supported in Chrome, but will **fail** starting in Chrome 58.

```
var widevineOptions = [
  { initDataTypes: [ 'cenc' ] }
];
navigator.requestMediaKeySystemAccess(
    'com.widevine.alpha', widevineOptions)
  .then(function(keySystemAccess) { ... } );
```

The following is currently supported and will continue to be **supported** in Chrome 58.

```
var widevineOptions = [
  {
    initDataTypes: [ 'cenc' ],
    audioCapabilities: [ {
      contentType: 'audio/mp4;codecs="mp4a.40.2"'
    } ],
    videoCapabilities: [ {
      contentType: 'video/mp4;codecs="avc1.4d401e"'
    } ]
  }
];
navigator.requestMediaKeySystemAccess(
    'com.widevine.alpha', widevineOptions)
  .then(function(keySystemAccess) { ... } );
```

This is only required for encrypted streams, but is good practice to set for all content.

## Secure Origin and Transport

In previous Chrome versions EME APIs were allowed on insecure content origins with only a warning message was shown in dev console:

> *requestMediaKeySystemAccess() is deprecated on insecure origins in the specification. Support will be removed in the future. You should consider switching your application to a secure origin, such as HTTPS. See https://goo.gl/rStTGz for more details.*

Public Intent to Remove: Insecure usage of EME

### What does the spec say?

Most APIs are only exposed to secure contexts [SECURE-CONTEXTS] as indicated by the [SecureContext] IDL attribute. For example:

```
partial interface Navigator {
    [SecureContext]
    Promise<MediaKeySystemAccess> requestMediaKeySystemAccess(
```

```
        DOMString keySystem,
        sequence<MediaKeySystemConfiguration> supportedConfigurations);
};
```

Also in https://w3c.github.io/encrypted-media/#privacy-secureorigin:

> *The APIs defined in this specification are only supported on secure origins, protecting information discussed in previous sections.*

### What will Chrome do starting from Chrome 58?

EME API will be disabled on insecure origins including content origins. This means the requestMediaKeySystemAccess() and other APIs will not be *visible* from insecure contexts as indicated by the `[SecureContext]` modifiers in the spec IDL. Only the “encrypted” and “waitingforkey” event handlers will remain exposed. This change should have no effect to applications that are currently using HTTPS for all origins including manifest and video/audio content.

### What should applications do?

Switch to use secure origin(s) before **April 2017**. Note that as is the case for all applications served from a secure origin, all XHRs and fetches (including media streams for MSE), all scripts <...> must also be served from a secure origin.  This specifically means DASH manifest and all video/audio content.

# Best Practices

## Avoid Empty Robustness

A highest robustness level(s) available on a given device, should always be specified for every license request. The application can specify multiple MediaKeySystemMediaCapability values with different robustness levels and make decisions based on which one is supported or simply specify one robustness level in the request.

EME Spec:
http://w3c.github.io/encrypted-media/#idl-def-mediakeysystemmediacapability

### What should applications do?

Always specify the robustness level(s) required (by the license server) and/or that would affect the streams selected. The application could specify multiple MediaKeySystemMediaCapability values with different robustness levels and make decisions based on which one is supported. Robustness levels should be set at the license proxy level and should be appropriate per platform and encryption of the content.

### Available Robustness Values

| Definition | EME Level | 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 |

## Client Compatibility

Support for Widevine robustness levels could vary on different platforms/devices. Also this could be affected by whether user gives explicit consent. In order to determine appropriate robustness, device capabilities can be queried on a given device.

The following is an example of checking device capabilities:

### Example

```
widevineOptions = [
  { label: 'foo', // For debugging and/or identifying the selected config.
    initDataTypes: ['cenc', 'webm'],
    sessionTypes: ['temporary'],
    audioCapabilities: [
      // Preferred codec.
      { contentType: 'audio/mp4; codecs="mp4a.40.5"',
        robustness: 'SW_SECURE_CRYPTO'
      },
      // Next most-preferred codec.
      { contentType: 'audio/mp4; codecs="mp4a.40.2"',
        robustness: 'SW_SECURE_CRYPTO'
      },
      // Least preferred codec.
      { contentType: 'audio/webm; codecs="vorbis"',
        robustness: 'SW_SECURE_CRYPTO'
      },
    ],
    videoCapabilities: [
      // Preferred codec.
      { contentType: 'video/webm; codecs="vp9"',
        robustness: 'HW_SECURE_ALL'
      },
      { contentType: 'video/webm; codecs="vp9"',
        robustness: 'SW_SECURE_DECODE'
      },
      // Next most-preferred codec.
      { contentType: 'video/mp4; codecs="avc1.640028"', // high profile
        robustness: 'HW_SECURE_ALL'
      },
      { contentType: 'video/mp4; codecs="avc1.640028"',
        robustness: 'SW_SECURE_DECODE'
      },
      // Least preferred codec.
      { contentType: 'video/mp4; codecs="avc1.4d401e"', // main profile
        robustness: 'HW_SECURE_ALL'
      },
      { contentType: 'video/mp4; codecs="avc1.4d401e"',
        robustness: 'SW_SECURE_DECODE'
      },
    ],
  }
];

navigator.requestMediaKeySystemAccess(
```

```
    'com.widevine.alpha', widevineOptions)
  .then(function(result) {
    keySystemAccess = result;
    config = keySystemAccess.getConfiguration();
    ...
  })
```

The returned values can be inspected to see what's supported and then the right stream can be selected to serve. For example, if HW_SECURE_ALL is supported with VP9, the application can serve L1 content encoded using VP9.  This can also be used in conjunction with Selecting a Supported Key System example in the spec to support multiple key systems.

It is important to note that selecting a robustness level that is not supported on a given device will result in a playback failure of encrypted content.