DRM Support in Vewd Core and devices

Getting Started: Adding DRM support to your application

1. Playing media: The HTMLMediaElement

The HTMLMediaElement interface adds to HTMLElement the properties and methods needed to support basic media-related capabilities that are common to audio and video. The HTMLVideoElement and HTMLAudioElement elements both inherit this interface. See https://developer.mozilla.org/en/docs/Web/API/HTMLMediaElement for more info.

2. Streaming media: Media Source Extensions (MSE):

We can load, decode and play media simply by providing a src URL:

```
<video src='foo.webm'></video>
```

The Media Source API is an extension to HTMLMediaElement enabling more fine-grained control over the source of media, by allowing JavaScript to build streams for playback from 'chunks' of video. This in turn enables techniques such as adaptive streaming and time shifting. Refer to https://www.w3.org/TR/media-source/ for the full specification.

3. Adaptive streaming of media: Dynamic Adaptive Streaming over HTTP (DASH)

The idea is to be able to adapt our video streams based on present circumstances, for example fluctuating bandwidth, etc. DASH allows us to dynamically adapt our video streams over time. This can be done in several ways, DASH is one of them. The 2 main other APIs are HTTP Live Streaming (HLS) developed by Apple and Microsoft Smooth Streaming (MSSS) developed by Microsoft. Both HLS and MSSS are proprietary, however, DASH is an open standard due to which it is the most popular (used in Youtube TV for example).

4. Encrypting your streamed media: Encrypted Media Extensions (EME)

This is where Digital Rights Management (DRM) starts to come in. Simply put, Encrypted Media Extensions (EME) provides an API that enables web applications to interact with content protection systems, to allow playback of encrypted audio and video. It is an extension to the HTMLMediaElement specification, hence the name. There are 2 types of EME support available: prefixed and un-prefixed. EME spec versions 0.1b and earlier are informally called prefixed EME while later implementations are called un-prefixed. This is because function calls in the older versions of the EME spec were prefixed with the browser vendor name (ex: msSetMediaKeys, webkitAddKey, etc) while the newer versions have a universal API which is browser-independent and thus un-prefixed.

Applications that use EME normally have the following components:

1. **A Keys System:** In order to decrypt the encrypted media, we need some kind of a key system. This is where DRM technologies come into play. EME does not mandate a particular key system, but does mandate that browsers implementing EME support implement Clear Key. The most popular Key systems are **Widevine** maintained by Google and **Playready** by Microsoft.
2. **A Content Decryption Module (CDM):** In order to be able to decrypt and ultimately play the video on device, a Content Decryption Module (CDM) must be present on the device. This can be in the form of a software or hardware implementation.
3. **A License Server:** The License server can communicate directly with the CDM to facilitate decryption of the encrypted media. However, all initial handshaking and communication is handled by the javascript application itself.

Here is a table with a list of the supported key systems for different Vewd Core and EME versions (taken from here):

TVSDK	EME 0.1b	Unprefixed EME (Draft 04 February 2016)
4.8.0	"webkit-org.w3.clearkey" "com.youtube.playready" "com.microsoft.playready" "com.widevine.alpha"	
4.9.0		"org.w3.clearkey" "com.youtube.playready" "com.microsoft.playready" "com.widevine.alpha"

Refer to this excellent article for a more thorough understanding of EME.

5. Choosing your Licensing Technology: DRM Backends

Now that you understand how EME works with a Keys system and a License server, it is time to select which licensing technology you want to use:

1. Microsoft Playready
2. Google Widevine

Vewd based browsers may support both PlayReady and Widevine, depending on capabilities of the platform integration. So your application needs to be tested on each device to make sure DRM playback works on that device. Unfortunately, our 4.x based Emulator does not provide PlayReady or Widevine support at the moment and we recommend testing your application on an Android based device (for example the NVidia Shield and Google Nexus TV) to test these out. Clear Key, however, is always supported internally by the Vewd Core and you can test it on any platform. This information is summarized in the table below:

	Clear Key	PlayReady	Widevine
Vewd TV Emulator 4.x	✓	✗ Not supported	✗ Not supported
Nvidia Shield/Nexus Player	✓	✓ Up to security level 2000	✓ Up to security level L1
Other devices	✓	✗ Dependent on device	✗ Dependent on device

Automatic Streaming & DRM Playback Methods

Manifest-based Streaming

Instead of manually initiating streaming and DRM playback methods (like getting keys, licenses, fetching video chunks, etc) in JavaScript, the Vewd Core provides the option of handling this automatically on your behalf for PlayReady videos. The idea is to pass in a manifest file which holds all the information about your audio/video content, and have Vewd automatically handle fetch, decode and playback. Devices that do have support for this feature, have support for the following manifest files:

- Microsoft Smooth Streaming (MSSS): *http://something/something*/**Manifest**
- Http Live Streaming (HLS): *http://something/something*.**m3u8**
- Dynamic Adaptive Streaming over HTTP (DASH): *http://something/something*.**mpd**

For example, in order to automatically playback a video via Microsoft Smooth Streaming, the following code should suffice:

```
<html>
<body>
<video width="512" height="288" controls autoplay>
  <source src="http://example.com/video/Manifest">
</video>
</body>
</html>
```

In addition to these streaming technologies, the Vewd Media Player Module also supports using PlayReady Web Initiator. This is described in the next section.

PlayReady Web Initiator

Devices that support the Vewd Media Player Module also support PlayReady Web Initiator, where a license acquisition manifest file is passed as the source of a <source> element. This manifest file in return contains a link to a Microsoft Smooth Streaming (MSSS) manifest file along with the license acquisition information. Below is a simple test case with 2 files: index.html and LicenseAcquisition.xml which can be used to test PlayReady Web Initiator support on your device:

index.html

```
 <html>
<body>
<video autoplay controls id="player" style="width: 100%">
    <source src="./LicenseAcquisition.xml" type="application/vnd.ms-playready.initiator+xml"/>
</video>
</body>
<script>
    var video = document.getElementById('player');
    video.onerror = function (err) {
        console.log(err);
    };
    video.onloadstart = function () {
        console.log('onloadstart');
    };
    video.onprogress = function () {
        console.log('progress')
    }
</script>
</html>
```

LicenseAcquisition.xml

```
<?xml version="1.0" encoding="utf-8"?>
<PlayReadyInitiator xmlns="http://schemas.microsoft.com/DRM/2007/03/protocols/">
<LicenseAcquisition>
    <Content>http://playready.directtaps.net/smoothstreaming/SSWSS720H264PR/SuperSpeedway_720.ism/Manifest<
/Content>
    <Header>
        <WRMHEADER xmlns="http://schemas.microsoft.com/DRM/2007/03/PlayReadyHeader" version="4.0.0.0">
        <DATA>
            <PROTECTINFO>
                <KEYLEN>16</KEYLEN>
                <ALGID>AESCTR</ALGID>
            </PROTECTINFO>
            <KID>AmfjCTOPbEOl3WD/5mcecA==</KID>
            <CHECKSUM>BGw1aYZ1YXM=</CHECKSUM>
            <CUSTOMATTRIBUTES>
                <IIS_DRM_VERSION>7.1.1064.0</IIS_DRM_VERSION>
            </CUSTOMATTRIBUTES>
            <LA_URL>http://playready.directtaps.net/pr/svc/rightsmanager.asmx</LA_URL>
            <DS_ID>AH+03juKbUGbHl1V/QIwRA==</DS_ID>
        </DATA>
    </WRMHEADER>
</Header>
</LicenseAcquisition>
</PlayReadyInitiator>
```

Troubleshooting

Here is a list of the common problems you might face when adding DRM support into your application:

	Problem	Possible Causes
1	License request is not generated by CDM	• DRM initialization data may not be correct or not supported by CDM
2	License request is rejected by server	• CDM may not support required security level or secure clock (PlayReady) • License server requires IP for specific location • License server requires additional information (ex for PlayReady CustomData)
3	License is rejected by CDM	• Not supported by CDM license (ex for PlayReady - secure clock is rewired but not supported)
4	License is acquired correctly but video doesn't play (no decode error)	• DRMBackend didn't send KeyStatusesChangeEvent with valid KID (ex. KID as GUID is wrong) • In JavaScript MediaKeys are not assigned to MediaElement • If other then MSE streaming is used then check if device supports the Vewd Media Player Module or Vewd Core version is older then 4.9.0. • If other then MSE streaming with PlayReady DRM is used then check if DRM initialization data in stream is correct (application may require automatic license acquisition)

More Information

- DCP EME tests: http://dcp.otvs.tv/ts/eme (ClearKey, PlayReady, Widevine)
- DCP DRM tests: http://dcp.otvs.tv/ts/drm (ClearKey, PlayReady, Widevine, PlayReady Web-Initiator)
- YouTube DASH player: http://yt-dash-mse-test.commondatastorage.googleapis.com/demo-player/dash-player-020416.html
- YouTube EME conformance: http://yt-dash-mse-test.commondatastorage.googleapis.com/unit-tests/2017.html
- Vewd Certify for Apps Specification: https://developer.vewd.com/display/OTV/Certify+for+Apps
- Vewd Certify for Devices Specification: https://developer.vewd.com/display/OTV/Certify+for+Devices
- DRM vendors: DRM Support in Vewd Core and devices, DRM Support in Vewd Core and devices