Click to prove you're human

Hls fmp4 test stream

Unified Streaming Table of Contents Note This part of our documentation is specifically intended for packaging to deliver fMP4 over HLS, or, in short: fMP4 HLS. This is a new feature and other parts of our documentation that refer to HLS without explicitly mentioning compatibility with fMP4 HLS may not necessarily apply to it, as these parts were written when Transport Streams (TS) were the only output format supported for delivery of HLS. Packaging for delivery of fMP4 HLS does not differ much from packaging HLS using Transport Streams. The most significant differences are the following: You package your content into a CMAF compatible container, instead of Transport Streams. See: How to package CMAF You never use the --package_hls option (not for creating the Media Playlists, I-frame Playlists or Master Playlist) A Media Playlist for fMP4 HLS uses byte ranges to index the CMAF packaged content. Therefore, no individual segments are generated You add a different kind of encryption to your content and use different options to do so Superficially, the resulting playlists look very similar. Do note though that regarding encryption, the exact meaning of SAMPLE-AES is different for fMP4 HLS than it is for HLS with Transport Streams. The first major benefit of fMP4 HLS is that it uses the same file container format as MPEG-DASH. You can read more about this convergence in our blog post about The promises of CMAF and its compatibility with the Unified Streaming Platform. In addition, Apple added support for HEVC and HDR (Dolby Vision and HDR10) to the HLS specification, but restricted these features to fMP4 HLS only. Using Unified Packager, no additional configuration is needed when packaging HEVC and/or HDR for fMP4 HLS. Below you can find the necessary steps for packaging audio and video content. Overall, they are the following: Make sure your content is packaged in a CMAF compatible container with the desired fragment duration (e.g., 8 seconds). See this short overview that shows How to package CMAF Create Media Playlists based on your CMAF packaged content Create I-frame Playlists for your video content Create a Master Playlist based on all the individual playlists that you created Adding encryption is done at the first step (when packaging your content as CMAF) and adding the necessary signaling for the encryption is done at the second step (when creating the Media Playlists). As mentioned above, the notable difference when creating the necessary playlists for streaming HLS from statically packaged content is that the --package_hls should not be used. Use tos-fmp4-hls-static.sh to package the Tears of Steel demo contents as CMAF and subsequently generate the necessary HLS playlists to set up a statically packaged fMP4 HLS stream. #!/bin/bash mp4split -o tos-64k-fmp4.m3u8 \ tos-8s-aac-64k.cmfa mp4split -o tos-128k-fmp4.m3u8 \ tos-8s-aac-128k.cmfa mp4split -o tos-400k-fmp4.m3u8 \ tos-8s-avc1-400k.cmfv mp4split -o tos-750k-fmp4.m3u8 \ tos-8s-avc1-750k.cmfv mp4split -o tos-1000k-fmp4.m3u8 \ tos-8s-avc1-1000k.cmfv Note It's also possible to create a HLS Media Playlist for subtitles that are stored in a fMP4 container (i.e., a .cmft-file). However, iOS and MacOS devices only support the TTML based variant of ISO-14496-30 (IMSC1 Text) in fMP4, not WebVTT. For WebVTT, only WebVTT Segments are supported. To generate those WebVTT Segments and the necessary Media Playlist, follow the instructions for Adding WebVTT subtitles when packaging HLS TS. In addition to creating standard Media Playlists for each track, create I-frame Playlists for every track to enable trick play and allow viewers to fast forward and rewind playback of your stream. To do so, you can: Create I-frame Playlists based on sync-samples only MP4s, as explained in Adding trick play to a DASH or HLS stream Or you can create I-frame Playlists based on the regular video files, by using the --create_iframe_playlist option The latter is explained below: #!/bin/bash mp4split -o iframe-tos-400k-fmp4.m3u8 \ --create_iframe_playlist \ tos-8s-avc1-400k.cmfv mp4split -o iframe-tos-750k-fmp4.m3u8 \ --create_iframe_playlist \ tos-8s-avc1-750k.cmfv mp4split -o iframe-tos-1000k-fmp4.m3u8 \ --create_iframe_playlist \ tos-8s-avc1-1000k.cmfv Note Also, the --create_iframe_playlist option used for fMP4 HLS should not be used for HLS with Transport Streams, as the necessary I-frame Playlists are generated automatically when packaging the latter. Packaging the media playlists into a Master Playlist is done the usual way. #!/bin/bash mp4split -o master_fmp4_clear.m3u8 \ tos-64k-fmp4.m3u8 \ tos-128k-fmp4.m3u8 \ tos-400k-fmp4.m3u8 \ tos-750k-fmp4.m3u8 \ tos-1000k-fmp4.m3u8 \ iframe-tos-400k-fmp4.m3u8 \ iframe-tos-750k-fmp4.m3u8 \ iframe-tos-1000k-fmp4.m3u8 When streaming fMP4 HLS, Apple specifies that the CENC 'cbcs' encryption scheme must be used if you want to protect your content. To do so, requires the following: Add 'cbcs' encryption when packaging your content as CMAF Add the required signaling of the keys when generating the Media Playlists Note When generating a Media Playlist, you can specify a value for the advisory HLS tag 'HDCP-LEVEL' using the option --hdcp_level, for which none, type-0 and type-1 are valid values (e.g., --hdcp_level=type-1). If this tag is not specified, it will not be added. To encrypt your content according to the CENC 'cbcs' encryption scheme you add the following parameters to your command-line when packaging your content to CMAF (with your own keys, of course). --cenc_key=100000001000100010001000000001:3a2a1b68dd2bd9b2eeb25e84c4776668 \ --cenc_key_iv=f9d9d5f5f3b5bc0372856df068f9f6a2 \ --scheme_type=cbcs Like so: #!/bin/bash CEK=3a2a1b68dd2bd9b2eeb25e84c4776668 KID=100000001000100010001000000001 KIV=f9d9d5f5f3b5bc0372856df068f9f6a2 mp4split -o tos-8s-128k-cbcs.cmfa \ --fragment_duration=384000/48000 \ --cenc_key=${KID}:${CEK} \ --cenc_key_iv=${KIV} \ --scheme_type=cbcs \ tears-of-steel-aac-128k.mp4 mp4split -o tos-8s-2200k-cbcs.cmfv \ --fragment_duration=192/24 \ --cenc_key=${KID}:${CEK} \ --cenc_key_iv=${KIV} \ --scheme_type=cbcs \ tears-of-steel-avc1-2200k.mp4 To add the necessary signaling of the key to your Media Playlists, make use of the --encrypt_key_url option when creating these playlists. For example: #!/bin/bash CEK=3a2a1b68dd2bd9b2eeb25e84c4776668 # Store binary representation of CEK in cek.key file echo ${CEK} | xxd -p -r > cek.key # URL that resolves to the cek.key file LA_URL= mp4split -o tos-128k-fmp4-cbcs.m3u8 \ --encrypt_key_url=${LA_URL} \ tos-8s-128k-cbcs.cmfa mp4split -o tos-2200k-fmp4-cbcs.m3u8 \ --encrypt_key_url=${LA_URL} \ tos-8s-2200k-cbcs.cmfv Then, with all the Media Playlists in place, you generate the I-frame Playlists and Master Playlist as you normally would. Use tos-fmp4-hls-cbcs-static.sh to package the Tears of Steel demo contents as CMAF with 'cbcs' encryption and subsequently generate the necessary HLS playlists to set up a statically packaged and encrypted fMP4 HLS stream. When you want to use FairPlay DRM, you have to set the --encrypt_key_url to signal the URL of your FairPlay licensing server. Additionally, you have to add the option --key_format="com.apple.streamingkeydelivery" when generating your Media Playlists. #!/bin/bash KID=100000001000100010001000000001 FP_URL=skd://${KID} mp4split -o tos-128k-fmp4-fp.m3u8 \ --encrypt_key_url=${FP_URL} \ --key_format="com.apple.streamingkeydelivery" \ tos-8s-128k-cbcs.cmfa mp4split -o tos-2200k-fmp4-fp.m3u8 \ --encrypt_key_url=${FP_URL} \ --key_format="com.apple.streamingkeydelivery" \ tos-8s-2200k-cbcs.cmfv Then, with all the Media Playlists in place, you generate the I-frame Playlists and Master Playlist as you normally would. To support PlayReady 4 for HLS, add the necessary signaling when generating the Media Playlists with Unified Packager, by using the --iss.drm_specific_data command-line option to specify the base64 encoded PlayReady Object containing the PlayReady Header (or, even better, the complete PSSH box): #!/bin/bash PR_OBJECT=uAIAAAEAAQCuAjwAVwBSAE0ASABFAEEARABFAFIAIAB4AG0AbABuAHMAPQAiAGgAdAB0AHAAOgAvAC8AcwBjAGgAZQBAGEAcwAuAG0AaQBjAHIAbwBzAG8AZgB0AC4AYwBvAG0ALwBEAFIATQAvADIAMAAwADcALwAwADMALwBQAGwAYQB5AFIAZQBhAGQAeQBIAGUAYQBkAGUAcgAiACAAdgBlAHIAcwBpAG8AbgA9ACIANAAuADMALgAwAC4AMAAiAD4APABEAEEAVABBAD4APABMAEEAXwBVAFIATAA+AGgAdAB0AHAAOgAvAC8AZQB4AHAAZQByAGkAbQBlAG4AdABhAGwAMQAuAGEAegB1AHIAZQB3AGUAYgBzAGkAdABlAHMALgBuAG
mp4split -o tos-128k-fmp4-pr.m3u8 \ --iss.drm_specific_data=${PR_OBJECT} \ tos-8s-128k-cbcs.cmfa mp4split -o tos-2200k-fmp4-pr.m3u8 \ --iss.drm_specific_data=${PR_OBJECT} \ tos-8s-2200k-cbcs.cmfv mp4split -o tos-master-pr.m3u8 \ tos-128k-fmp4-pr.m3u8 \ tos-2200k-fmp4-pr.m3u8 To support Widevine DRM for HLS V2, add the necessary signaling when generating the Media Playlists with Unified Packager, by using the --widevine.drm_specific_data command-line option to specify the base64 encoded Widevine data (or, even better, the complete PSSH box). #!/bin/bash WV_DATA=IhBma2ozbGphU2RmYWxrcjNqSOPclZsG mp4split -o tos-128k-fmp4-wv.m3u8 \ --widevine.drm_specific_data=${WV_DATA} \ tos-8s-128k-cbcs.cmfa mp4split -o tos-2200k-fmp4-wv.m3u8 \ --widevine.drm_specific_data=${WV_DATA} \ tos-8s-2200k-cbcs.cmfv mp4split -o tos-master-wv.m3u8 \ tos-128k-fmp4-wv.m3u8 \ tos-2200k-fmp4-wv.m3u8 © Copyright 2007-2025, Unified Streaming. Built with Sphinx using a theme provided by Read the Docs. When you're working with HLS, whether you're testing an HLS player or just trying to figure out how HLS works, it's super handy to have some sample HLS m3u8 URLs ready to test. I often catch myself googling for these URLs, so I've put together a list for you right here! VOD HLS Stream 1. Becoming You 2. Skate Phantom Flex 3. Tears of Steel 4. Tears of Steel 5. Big Buck Bunny 6. DAI Discontinuity (Ad-insertion in event stream) 7. Historic Planet 3D Test HLS Stream for Apple Vision Pro (visionOS) 8. Bip-Bop 9. Sintel LIVE HLS Stream 1. Tears of Steel 2. Sky News Portrait HLS Video Other resources Azure Media Services Sample Streams Note - We don't own these sample video streams, and they might stop working at any time. HLSHLS DebugHLS TestingHLS StreamHLS Stream Sample This demo will work on Mac, iOS 10 and above, and for PC users in all recent browser versions of e.g., Edge, Firefox, and Chrome. const conf = { key: " }; var source = { hls: ', poster: ' }; var playerContainer = document.getElementById('player-container'); var player = new bitmovin.player.Player(playerContainer, conf); player.load(source); Find out more about HLS with fragmented MP4 in our blog post. Access your free account today by signing up for a trial At WWDC 2016, Apple announced support for fragmented MP4 (fMP4) as an alternative to MPEG-TS, which prior to their announcement was the only supported format. So why use fragmented MP4 files? Well, according to Apple's video encoding requirements in their HLS Authoring Specification, if you want to use HEVC/H.265, you have to use it (1.5). Fragmented MP4 files are also compatible with MPEG-DASH – an alternative to HLS – so you can use the same files; only the manifest file (playlist) is different. This means less encoding and less storage requirements, which should reduce costs. In this post, I'll demonstrate how to generate fMP4 files using ffmpeg. You'll need a fairly recent version of ffmpeg that supports fMP4 – I used version 4.1.1 to create the files in this post. For the video, I used the 1080p version of the Sintel trailer. You can download it here. Let's take the original video (sintel_trailer-1080p.mp4) and re-encode it. We'll alter the size to 1280×720 and reduce the video bitrate but leave the audio as it is. Open up a terminal and run the following command: ffmpeg -y \ -i sintel_trailer-1080p.mp4 \ -force_key_frames "expr:gte(t,n_forced*2)" \ -sc_threshold 0 \ -s 1280x720 \ -c:v libx264 -b:v 1500k \ -c:a copy \ -hls_time 6 \ -hls_playlist_type vod \ -hls_segment_type fmp4 \ -hls_segment_filename "fileSequence%d.m4s" \ prog_index.m3u8 To generate fMP4 files instead of MPEG-TS, which is the default, set the segment type to fmp4 (highlighted). The above command will generate the following files: the playlist file (prog_index.m3u8), a number of files with the extension .m4s (instead of .ts), and a file called init.mp4. You'll need all of these files when you come to play the video. Let's take a look at the playlist and see how it compares: #EXTM3U #EXT-X-VERSION:7 #EXT-X-TARGETDURATION:6 #EXT-X-MEDIA-SEQUENCE:0 #EXT-X-PLAYLIST-TYPE:VOD #EXT-X-MAP:URI="init.mp4" #EXTINF:6.000000, fileSequence0.m4s #EXTINF:6.000000, fileSequence1.m4s #EXTINF:6.000000, fileSequence2.m4s #EXTINF:6.000000, fileSequence3.m4s #EXTINF:6.000000, fileSequence4.m4s #EXTINF:6.000000, fileSequence5.m4s #EXTINF:6.000000, fileSequence6.m4s #EXTINF:6.000000, fileSequence7.m4s #EXTINF:4.208333, fileSequence8.m4s #EXT-X-ENDLIST I've highlighted the differences. The #EXT-X-MAP tag specifies the location of the resource that contains the information required to parse the segments (the media initialization section). It applies to every segment that appears after it. A playlist containing an MPEG-2 transport stream can also apply the #EXT-X-MAP tag, but each segment typically includes the information required to parse it so it's not needed. The other difference is the file extension of the segments is .m4s instead of .ts. HLS also supports byte range addressing. To use this feature with fMP4, set the -hls_flags option to single_file. Instead of creating several (smaller) files, it will create a single file containing all the media segments. If you look at the playlist, the location of each segment is specified as a byte offset in the file. Now that we have the fMP4 files and the playlist, the next step is to play the video in the browser. Most browsers don't support HLS and fMP4 natively so you'll need to use a video player that utilises Media Source Extensions (MSE). Thankfully, some players do but be aware that some don't. For example, I tried using hls.js but it didn't work – according to the documentation, support for fragmented MP4 is still in beta. Here's an HTML example that uses Video.js, which does support fMP4: