Advances in HTTP Live Streaming

Session 504

Roger Pantos, AVFoundation Engineer
Anil Katti, AVFoundation Engineer
But First
HLS Has Been Approved for Publication as an RFC!

draft-pantos-http-live-streaming-23 will be published by the IETF as an RFC
HLS Has Been Approved for Publication as an RFC!

draft-pantos-http-live-streaming-23 will be published by the IETF as an RFC
Once it moves through the publication queue it will be assigned an RFC number
HLS Has Been Approved for Publication as an RFC!

draft-pantos-http-live-streaming-23 will be published by the IETF as an RFC.
Once it moves through the publication queue it will be assigned an RFC number.
At that point, it will serve as a stable reference for HLS.
HLS Has Been Approved for Publication as an RFC!

draft-pantos-http-live-streaming-23 will be published by the IETF as an RFC

Once it moves through the publication queue it will be assigned an RFC number

At that point, it will serve as a stable reference for HLS

Watch for a new Internet-Draft containing future updates
New Video Format: HEVC
Benefits of HEVC

Better encoding efficiency than AVC / H.264
• Reduce segment sizes by up to 40% with same visual quality

On a given network link, this translates to:
• Faster video start at reasonable quality
• Better quality overall
### HEVC Will Be Widely Supported

<table>
<thead>
<tr>
<th>Feature</th>
<th>iOS</th>
<th>macOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-bit Hardware Decode</td>
<td>A9 chip</td>
<td>6th Generation Intel Core processor</td>
</tr>
<tr>
<td>(includes FairPlay Streaming)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10-bit Hardware Decode</td>
<td></td>
<td>7th Generation Intel Core processor</td>
</tr>
<tr>
<td>(includes FairPlay Streaming)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8-bit Software Decode</td>
<td>All iOS Devices</td>
<td>All Macs</td>
</tr>
<tr>
<td>10-bit Software Decode</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
HEVC Support in HLS

For Apple clients, HEVC must be packaged as fMP4
• No support for HEVC in MPEG-2 TS
HEVC Support in HLS

For Apple clients, HEVC must be packaged as fMP4
• No support for HEVC in MPEG-2 TS

Same encryption format — 'cbcs' ISO/IEC 23001:7 Common Encryption
HEVC Support in HLS

For Apple clients, HEVC must be packaged as fMP4

• No support for HEVC in MPEG-2 TS

Same encryption format — 'cbcs' ISO/IEC 23001:7 Common Encryption

Mark your Media Playlists as HEVC with the CODECS attribute

• CODECS= "hvc1.2.4.L123.B0,..."

• See ISO/IEC 14496-15 for the string format
Mixing HEVC and H.264

HEVC and H.264 variants can appear in the same Master playlist
• I-frame playlists too
Mixing HEVC and H.264

HEVC and H.264 variants can appear in the same Master playlist
• I-frame playlists too
• HEVC must be fMP4
• H.264 can be TS or fMP4
Mixing HEVC and H.264

HEVC and H.264 variants can appear in the same Master playlist
• I-frame playlists too
• HEVC must be fMP4
• H.264 can be TS or fMP4
• Label your Media Playlists with correct CODECS attributes!
Mixing HEVC and H.264

HEVC and H.264 variants can appear in the same Master playlist
• I-frame playlists too
• HEVC must be fMP4
• H.264 can be TS or fMP4
• Label your Media Playlists with correct CODECS attributes!

HLS Authoring guidelines have been updated for HEVC
• See the on-demand talk "HLS Authoring Update"
New Subtitle Format: IMSC1
New Subtitle Format: IMSC1

IMSC1 is a profile of TTML that has been optimized for streaming delivery
New Subtitle Format: IMSC1

IMSC1 is a profile of TTML that has been optimized for streaming delivery.

Better control over styling, compared to VTT.
New Subtitle Format: IMSC1

IMSC1 is a profile of TTML that has been optimized for streaming delivery.

Better control over styling, compared to VTT.

Baseline subtitle format for the Common Media Application Format (CMAF)

• Part of a wider effort to support CMAF features.
IMSC1 in HLS

Carried as XML text inside fMP4 Segments
IMSC1 in HLS

Carried as XML text inside fMP4 Segments

HLS defines support for the IMSC1 Text profile
IMSC1 in HLS

Carried as XML text inside fMP4 Segments

HLS defines support for the IMSC1 Text profile

Mark your Media Playlists as IMSC1 with the CODECS attribute

• CODECS="stpp.TTML.im1t, ..."
IMSC1 in HLS

Carried as XML text inside fMP4 Segments

HLS defines support for the IMSC1 Text profile

Mark your Media Playlists as IMSC1 with the CODECS attribute

- `CODECS="stpp.TTML.imlt, ..."`

Note that IMSC1 does not depend on HEVC, or vice-versa
//IMSC1 alongside WebVTT in a Master Playlist

#EXTM3U

#EXT-X-MEDIA:TYPE=SUBTITLES,GROUP-ID="vtt",LANGUAGE="eng",NAME="English",URI="vtt.m3u8"
#EXT-X-STREAM-INF:BANDWIDTH=90000,CODECS="avc1.4d001e,ac-3",SUBTITLES="vtt"
bipbop_gear1/prog_index.m3u8

#EXT-X-MEDIA:TYPE=SUBTITLES,GROUP-ID="imsc",LANGUAGE="eng",NAME="English",URI="imsc.m3u8"
#EXT-X-STREAM-INF:BANDWIDTH=90000,CODECS="avc1.4d001e,ac-3,stpp.TTML.im1t",SUBTITLES="imsc"
bipbop_gear1/prog_index.m3u8
IMSC1 in a Media Playlist

```markdown
# WebVTT
#EXTM3U
#EXT-X-TARGETDURATION:6
#EXTINF 6,
segment1.vtt
#EXTINF 6,
segment2.vtt
...
```

```markdown
# IMSC1
#EXTM3U
#EXT-X-TARGETDURATION:6
#EXT-X-MAP:URI="header.mp4"
#EXTINF 6,
segment1.mp4
#EXTINF 6,
segment2.mp4
...
```
You Might Switch Your HLS Streams to IMSC1 if ...

You want more control over text styling than VTT alone provides
You Might Switch Your HLS Streams to IMSC1 if ...

You want more control over text styling than VTT alone provides

You produce VTT by translating TTML

• TTML-to-IMSC1 translation is simpler, and may have higher fidelity
You Might Switch Your HLS Streams to IMSC1 if ...

You want more control over text styling than VTT alone provides

You produce VTT by translating TTML
  • TTML-to-IMSC1 translation is simpler, and may have higher fidelity

You produce IMSC1 anyway
  • Reduce the number of overall streams you produce
You Might Switch Your HLS Streams to IMSC1 if ...

You want more control over text styling than VTT alone provides

You produce VTT by translating TTML
• TTML-to-IMSC1 translation is simpler, and may have higher fidelity

You produce IMSC1 anyway
• Reduce the number of overall streams you produce

Sticking with VTT is fine, too
Is There an IMSC2?

Not yet. It is currently being defined.
Is There an IMSC2?

Not yet. It is currently being defined.

We expect it to add advanced styling features for Japanese text
Is There an IMSC2?

Not yet. It is currently being defined.

We expect it to add advanced styling features for Japanese text

Stay tuned
New Streaming Features
EXT–X–GAP: A New m3u8 Tag

#EXTM3U
#EXT–X–TARGETDURATION:10
#EXT–X–VERSION:7
#EXT–X–MEDIA–SEQUENCE:9
#EXTINF:9.34267,
fileSequence9.ts
#EXTINF:9.75975,
fileSequence10.ts
#EXTINF:9.63462,
fileSequence11.ts
#EXTINF:9.34267,
fileSequence12.ts
EXT–X–GAP: A New m3u8 Tag

#EXTM3U
#EXT-X-TARGETDURATION:10
#EXT-X-VERSION:7
#EXT-X-MEDIA-SEQUENCE:10
#EXTINF:9.75975,
fileSequence10.ts
#EXTINF:9.63462,
fileSequence11.ts
#EXTINF:9.34267,
fileSequence12.ts
#EXTINF:9.75975,
fileSequence13.ts

EXT-X-GAP: A New m3u8 Tag
EXT-X-GAP: A New m3u8 Tag

#EXTM3U
#EXT-X-TARGETDURATION:10
#EXT-X-VERSION:7
#EXT-X-MEDIA-SEQUENCE:11
#EXTINF:9.63462,
fileSequence11.ts
#EXTINF:9.34267,
fileSequence12.ts
#EXTINF:9.75975,
fileSequence13.ts
#EXTINF:9.84317,
fileSequence14.ts
EXT-X-GAP: A New m3u8 Tag

#EXTM3U
#EXT-X-TARGETDURATION:10
#EXT-X-VERSION:7
#EXT-X-MEDIA-SEQUENCE:12
#EXTINF:9.34267,
fileSequence12.ts
#EXTINF:9.75975,
fileSequence13.ts
#EXTINF:9.84317,
fileSequence14.ts
#EXTINF:8.75875,
#EXT-X-GAP
missing-Sequence15.ts

EXT-X-GAP: A New m3u8 Tag

#EXTM3U
#EXT-X-TARGETDURATION:10
#EXT-X-VERSION:7
#EXT-X-MEDIA-SEQUENCE:12
#EXTINF:9.34267,
fileSequence12.ts
#EXTINF:9.75975,
fileSequence13.ts
#EXTINF:9.84317,
fileSequence14.ts
#EXTINF:8.75875,
#EXT-X-GAP
missing-Sequence15.ts
EXT-X-GAP: A New m3u8 Tag

#EXTM3U
#EXT-X-TARGETDURATION:10
#EXT-X-VERSION:7
#EXT-X-MEDIA-SEQUENCE:13
#EXTINF:9.75975,
fileSequence13.ts
#EXTINF:9.84317,
fileSequence14.ts
#EXTINF:8.75875,
#EXT-X-GAP
missing-Sequence15.ts
#EXTINF:9.88487,
#EXT-X-GAP
missing-Sequence16.ts
EXT–X–GAP: A New m3u8 Tag

#EXTM3U
#EXT-X-TARGETDURATION:10
#EXT-X-VERSION:7
#EXT-X-MEDIA-SEQUENCE:14
#EXTINF:9.84317,
fileSequence14.ts
#EXTINF:8.75875,
#EXT-X-GAP
missing-Sequence15.ts
#EXTINF:9.88487,
#EXT-X-GAP
missing-Sequence16.ts
#EXTINF:9.09242,
fileSequence17.ts
EXT-X-GAP: A New m3u8 Tag

#EXTM3U
#EXT-X-TARGETDURATION:10
#EXT-X-VERSION:7
#EXT-X-MEDIA-SEQUENCE:15
#EXTINF:8.75875,
#EXT-X-GAP
missing-Sequence15.ts
#EXTINF:9.88487,
#EXT-X-GAP
missing-Sequence16.ts
#EXTINF:9.09242,
fileSequence17.ts
#EXTINF:9.63462,
fileSequence18.ts
EXT-X-GAP Tag Semantics

Gap tag indicates that a Media Segment is missing
EXT-X-GAP Tag Semantics

Gap tag indicates that a Media Segment is missing

• Player will not attempt to download the segment URL
EXT-X-GAP Tag Semantics

Gap tag indicates that a Media Segment is missing

- Player will not attempt to download the segment URL
- Player will attempt to find another variant without a gap to play
EXT-X-GAP Tag Semantics

Gap tag indicates that a Media Segment is missing

- Player will not attempt to download the segment URL
- Player will attempt to find another variant without a gap to play
- If all variants have the same gap, silence will be played until gap ends
EXT-X-GAP Tag Semantics

Gap tag indicates that a Media Segment is missing
• Player will not attempt to download the segment URL
• Player will attempt to find another variant without a gap to play
• If all variants have the same gap, silence will be played until gap ends

See the WWDC HLS beta spec for details
Variable Support in m3u8 Playlists

Simple variable substitution in m3u8 playlists
Variable Support in m3u8 Playlists

Simple variable substitution in m3u8 playlists

PHP-style syntax for variables: `${filename}.ts`
Variable Support in m3u8 Playlists

Simple variable substitution in m3u8 playlists

PHP-style syntax for variables: `{$filename}.ts`

`EXT-X-DEFINE` tag defines a variable
- Or imports it from master playlist

All variables must be defined, or playlist will not parse
Variable Support in m3u8 Playlists

Simple variable substitution in m3u8 playlists

PHP-style syntax for variables: `{filename}.ts`

`EXT-X-DEFINE` tag defines a variable

- Or imports it from master playlist

All variables must be defined, or playlist will not parse

Allows media playlists to depend on values defined in master playlist

- Media Playlist must explicitly import each variable
#EXTM3U

#EXT-X-DEFINE:NAME="auth",VALUE="?auth_token=/aazv/54334:pp2"

#EXT-X-STREAM-INF:BANDWIDTH=1156000,RESOLUTION=640x480,CODECS="avc1.4d001e,mp4a.40.2"
bipbop_gear1/prog_index.m3u8{$auth}
Variables in Media Playlists

#EXTM3U
#EXT-X-TARGETDURATION:6

#EXT-X-DEFINE:NAME="path",VALUE="/media/encoded/asset127-a/1MB/"
#EXT-X-DEFINE:IMPORT="auth"

#EXT-X-MEDIA-SEQUENCE:44
#EXTINF 6,
{$path}segment44.mp4{$auth}

See the WWDC HLS beta spec for details
Synchronized Playback of Live Streams

Playback is synchronized using shared `EXT-X-PROGRAM-DATE-TIME` tags
Synchronized Playback of Live Streams

Playback is synchronized using shared EXT-X-PROGRAM-DATE-TIME tags.

Use AVPlayer setRate:time:atHostTime: to start second player in sync.
Synchronized Playback of Live Streams

Playback is synchronized using shared EXT-X-PROGRAM-DATE-TIME tags

Use `[AVPlayer setRate:time:atHostTime:]` to start second player in sync

• Sample code available, "SyncStartTV"
Demo

Synchronized live stream playback
Resolution Cap: preferredMaximumResolution

Companion to existing bandwidth cap (preferredPeakBitRate)
Resolution Cap: preferredMaximumResolution

Companion to existing bandwidth cap (preferredPeakBitRate)

Programmatically specify the maximum desired content resolution
Resolution Cap: preferredMaximumResolution

Companion to existing bandwidth cap (preferredPeakBitRate)

Programmatically specify the maximum desired content resolution

Useful for video thumbnails and multi-stream presentations
Resolution Cap: preferredMaximumResolution

Companion to existing bandwidth cap (preferredPeakBitRate)

Programmatically specify the maximum desired content resolution

Useful for video thumbnails and multi-stream presentations

If there is no playable variant below the resolution cap, the lowest-resolution variant is chosen
func configureMyPlayerItem(item: AVPlayerItem) {
    item.preferredMaximumResolution = CGSize(height: 640, width: 480)
    
    
}
HLS Offline Storage Management
Netfix
Version 9.20.0

App Size
118 MB

Documents & Data
925.5 MB

Offload App
This will free up storage used by the app, but keep its documents and data. Reinstalling the app will place back your data if the app is still available in the App Store.

Delete App
This will delete the app and all related data from this iPhone. This action can't be undone.

DOWNLOADED VIDEOS

Dave Chappelle: C1 - The Age...
Viewed on May 31, 2017
326.7 MB

Master of None: S2 - Le Nozze
Viewed on May 31, 2017
156.8 MB

Orange Is the New Black: S2 - ...
Never viewed
309.8 MB
### Netflix

**Version**: 9.20.0

**App Size**: 118 MB

**Documents & Data**: 925.5 MB

---

**Offload App**

This will free up storage used by the app, but keep its documents and data. Reinstalling the app will place back your data if the app is still available in the App Store.

**Delete App**

This will delete the app and all related data from this iPhone. This action can't be undone.

---

**Downloaded Videos**

- **Dave Chappelle: C1 - The Age**
  - Viewed on May 31, 2017
  - 326.7 MB

- **Master of None: S2 - Le Nozze**
  - Viewed on May 31, 2017
  - 156.8 MB

- **Orange Is the New Black: S2 - Le Nozze**
  - Never viewed
  - 309.8 MB
The OS can delete offline assets while the app is not running

• Via Settings, or when space is required for a software update
HLS Offline Storage Management

The OS can delete offline assets while the app is not running
• Via Settings, or when space is required for a software update

AVAssetDownloadStorageManager
HLS Offline Storage Management

The OS can delete offline assets while the app is not running
• Via Settings, or when space is required for a software update

AVAssetDownloadStorageManager

Sets the policy for automatic purging of downloaded AVAssets
HLS Offline Storage Management

The OS can delete offline assets while the app is not running
• Via Settings, or when space is required for a software update

AVAssetDownloadStorageManager

Sets the policy for automatic purging of downloaded AVAssets
• AVAssetDownloadStorageManagementPolicy has two properties:
HLS Offline Storage Management

The OS can delete offline assets while the app is not running
• Via Settings, or when space is required for a software update

AVAssetDownloadStorageManager

Sets the policy for automatic purging of downloaded AVAssets
• AVAssetDownloadStorageManagementPolicy has two properties:
  - Expiration date
HLS Offline Storage Management

The OS can delete offline assets while the app is not running
• Via Settings, or when space is required for a software update

AVAssetDownloadStorageManager

Sets the policy for automatic purging of downloaded AVAssets
• AVAssetDownloadStorageManagementPolicy has two properties:
  - Expiration date
  - Priority (important, default)
// AVAssetDownloadStorageManager

// Get the singleton
let storageManager = AVAssetDownloadStorageManager.shared()
// AVAssetDownloadStorageManager

// Get the singleton
let storageManager = AVAssetDownloadStorageManager.shared()

// Set the policy
let newPolicy = AVMutableAssetDownloadStorageManagementPolicy()
newPolicy.expirationDate = myExpiryDate
newPolicy.priority = .important
storageManager.setStorageManagementPolicy(newPolicy, forURL: myDownloadStorageURL)
let storageManager = AVAssetDownloadStorageManager.shared()

let newPolicy = AVMutableAssetDownloadStorageManagementPolicy()
newPolicy.expirationDate = myExpiryDate
newPolicy.priority = .important
storageManager.setStorageManagementPolicy(newPolicy, forURL: myDownloadStorageURL)

let currentPolicy = storageManager.storageManagementPolicy(forURL: myDownloadStorageURL)
Batching up Your Offline Downloads

AVAggregateAssetDownloadTask

• Specify multiple media selections prior to initiating download
AVAggregateAssetDownloadTask

• Specify multiple media selections prior to initiating download

```swift
let task = myDownloadSession.aggregateAssetDownloadTask(with: AVURLAsset,
mediaSelections: AVMediaSelection[],
assetTitle: String,
assetArtworkData: Data?,
options: [String: Any]?)
```
Key Management Enhancements
FairPlay Streaming
FairPlay Streaming

FairPlay Streaming introduced in 2015
FairPlay Streaming

FairPlay Streaming introduced in 2015

Protects HLS content
FairPlay Streaming

FairPlay Streaming introduced in 2015

Protects HLS content

Enhancements to
• Simplify workflow
FairPlay Streaming

FairPlay Streaming introduced in 2015

Protects HLS content

Enhancements to
  • Simplify workflow
  • Scale adoption
FairPlay Streaming introduced in 2015

Protects HLS content

Enhancements to
- Simplify workflow
- Scale adoption
- Support new features
FairPlay Streaming Overview
FairPlay Streaming Overview

Securely deliver Content Decryption Keys
FairPlay Streaming Overview

Media-Centric Application
FairPlay Streaming Overview

Media-Centric Application

FPS Key Server Module
FairPlay Streaming Overview

- Media-Centric Application
- AVFoundation
- FPS Key Server Module
FairPlay Streaming Overview

1. Request Playback

- Media-Centric Application
- AVFoundation
- FPS Key Server Module
1. Request Playback
2. Request key
FairPlay Streaming Overview

1. Request Playback
2. Request key
3. Create SPC
FairPlay Streaming Overview

1. Request Playback
2. Request key
3. Create SPC
4. Send SPC; Get CKC
1. Request Playback
2. Request key
3. Create SPC
4. Send SPC; Get CKC
5. Respond with CKC

FairPlay Streaming Overview

AVFoundation Delegate

AVFoundation

 FPS Key Server Module
FairPlay Streaming Keys

- Media-Centric Application
- AVFoundation Delegate
- AVFoundation
FairPlay Streaming Keys

- Media-Centric Application
  - AVFoundation Delegate

- Playlists
- Media Data
FairPlay Streaming Keys

- Media-Centric Application
  - AVFoundation Delegate
- Start Playback
- AVFoundation
- FPS Keys
- Playlists
- Media Data
FairPlay Streaming Keys

- Media-Centric Application
  - AVFoundation Delegate
- Start Playback
- AVFoundation

- FPS Keys
- Playlists
- Media Data
FairPlay Streaming Keys

Media-Centric Application

AVFoundation Delegate

Start Playback

AVFoundation

FPS Keys

Playlists

Media Data

AVAssetResourceLoader
FairPlay Streaming Keys
FPS keys are more specialized resources
FairPlay Streaming Keys

FPS keys are more specialized resources

• Freeze dry for future use (offline playback)
FairPlay Streaming Keys

FPS keys are more specialized resources

• Freeze dry for future use (offline playback)
• Set up to expire - need renewal
FairPlay Streaming Keys

FPS keys are more specialized resources

• Freeze dry for future use (offline playback)
• Set up to expire - need renewal
• More such operations as content protection evolves
FairPlay Streaming Keys

FPS keys are more specialized resources
• Freeze dry for future use (offline playback)
• Set up to expire - need renewal
• More such operations as content protection evolves

FPS keys can be loaded independently of assets
FairPlay Streaming Keys

FPS keys are more specialized resources
• Freeze dry for future use (offline playback)
• Set up to expire - need renewal
• More such operations as content protection evolves

FPS keys can be loaded independently of assets

New API to manage and deliver decryption keys
FairPlay Streaming Keys

FPS keys are more specialized resources
• Freeze dry for future use (offline playback)
• Set up to expire - need renewal
• More such operations as content protection evolves

FPS keys can be loaded independently of assets

New API to manage and deliver decryption keys

AVContentKeySession
AVContentKeySession
AVContentKeySession

AVFoundation class for decryption keys
AVContentKeySession

AVFoundation class for decryption keys

• Decouples key loading from media loading or playback
AVContentKeySession

AVFoundation class for decryption keys
• Decouples key loading from media loading or playback
• Better control over lifecycle of keys
AVContentKeySession

AVFoundation class for decryption keys

• Decouples key loading from media loading or playback
• Better control over lifecycle of keys

Allows you to load keys at any time
AVFoundation class for decryption keys

- Decouples key loading from media loading or playback
- Better control over lifecycle of keys

Allows you to load keys at any time

Two ways key loading process is triggered:

AVContentKeySession
AVContentKeySession

AVFoundation class for decryption keys
• Decouples key loading from media loading or playback
• Better control over lifecycle of keys

Allows you to load keys at any time

Two ways key loading process is triggered:

Application
Using AVContentKeySession
AVContentKeySession

AVFoundation class for decryption keys

• Decouples key loading from media loading or playback
• Better control over lifecycle of keys

Allows you to load keys at any time

Two ways key loading process is triggered:

- **Application**
  - Using AVContentKeySession

- **AVFoundation**
  - After playback starts (content is encrypted)
Improve Playback Startup
Improve Playback Startup

Key load time can significantly impact startup
Improve Playback Startup

Key load time can significantly impact startup

Keys are normally loaded on-demand
Improve Playback Startup

Key load time can significantly impact startup

Keys are normally loaded on-demand

Use `AVContentKeySession` to

- Predictively load keys before requesting playback (key preloading)
Scale Live Playback
Scale Live Playback

Huge growth in live content
Scale Live Playback

Huge growth in live content

Extra protection with key rotation and lease renewal
Scale Live Playback

Huge growth in live content

Extra protection with key rotation and lease renewal

Impulse load on key server
Scale Live Playback

Huge growth in live content

Extra protection with key rotation and lease renewal

Impulse load on key server

Use `AVContentKeySession` to

• Load balance key requests at the point of origin
Scale Live Playback

Key Server
Scale Live Playback

Key Server

Multiple Users
Watching a Live Stream

Initial
Key Request
Scale Live Playback

Key Server

New Key Request
(Rotation / Renewal)
Scale Live Playback

Key Server
Scale Live Playback
Scale Live Playback

Key Server

New Key Request
(Rotation / Renewal)
Scale Live Playback

Key Server
Scale Live Playback

Key Server
Scale Live Playback

Key Server

New Key Request
(Rotation / Renewal)
Scale Live Playback

Key Server
Scale Live Playback

Key Server
Scale Live Playback

Key Server
//Using AVContentKeySession to initiate Key Loading Process

//Create AVContentKeySession instance for FairPlay Streaming Key Delivery
let contentKeySession = AVContentKeySession(keySystem: .fairPlayStreaming)

//Register self as Delegate
contentKeySession.setDelegate(self, queue: DispatchQueue(label: "DelegateQueue"))

//Initiate Key Loading Process
contentKeySession.processContentKeyRequest(withIdentifier: "skd://myKey", initializationData: nil, options: nil)
//Using AVContentKeySession to initiate Key Loading Process

//Create AVContentKeySession instance for FairPlay Streaming Key Delivery
let contentKeySession = AVContentKeySession(keySystem: .fairPlayStreaming)

//Register self as Delegate
contentKeySession.setDelegate(self, queue: DispatchQueue(label: "DelegateQueue"))

//Initiate Key Loading Process
contentKeySession.processContentKeyRequest(withIdentifier: "skd://myKey", initializationData: nil, options: nil)
//Using AVContentKeySession to initiate Key Loading Process

//Create AVContentKeySession instance for FairPlay Streaming Key Delivery
let contentKeySession = AVContentKeySession(keySystem: .fairPlayStreaming)

//Register self as Delegate
contentKeySession.setDelegate(self, queue: DispatchQueue(label: "DelegateQueue"))

//Initiate Key Loading Process
contentKeySession.processContentKeyRequest(withIdentifier: "skd://myKey", initializationData: nil, options: nil)
//Using AVContentKeySession to initiate Key Loading Process

//Create AVContentKeySession instance for FairPlay Streaming Key Delivery
let contentKeySession = AVContentKeySession(keySystem: .fairPlayStreaming)

//Register self as Delegate
contentKeySession.setDelegate(self, queue: DispatchQueue(label: "DelegateQueue"))

//Initiate Key Loading Process
contentKeySession.processContentKeyRequest(withIdentifier: "skd://myKey", initializationData: nil, options: nil)

//Tag in your Media Playlist
#EXT-X-KEY:METHOD=SAMPLE-AES,URI="skd://myKey",KEYFORMAT="com.apple.streamingkeydelivery",KEYFORMATVERSIONS="1"
Delegate callback that delivers AVContentKeyRequest

```swift
func contentKeySession(_ session: AVContentKeySession, didProvide keyRequest: AVContentKeyRequest) {
}
```
//Delegate callback that delivers AVContentKeyRequest
func contentKeySession(_ session: AVContentKeySession, didProvide keyRequest: AVContentKeyRequest) {

Delegate callback that delivers AVContentKeyRequest

```swift
func contentKeySession(_ session: AVContentKeySession, didProvide keyRequest: AVContentKeyRequest) {
    //Request SPC
    keyRequest.makeStreamingContentKeyRequestData(forApp: appCertificate, contentIdentifier: assetID, options: keyRequestOptions) {
        (spcBytes: Data?, spcCreationError: Error?) in
        //Send SPC to Key Server and obtain CKC
        if let ckcBytes = getCKCFromKeyServer(forSPC: spcBytes) {
            //Create Response using the CKC you obtained from Key Server
            let response = AVContentKeyResponse(fairPlayStreamingKeyResponseData: ckcBytes)
            //Set Response on the Key Request object when you are about to start playback
            keyRequest.processContentKeyResponse(response)
        }
    }
}
```
func contentKeySession(_ session: AVContentKeySession, didProvide keyRequest: AVContentKeyRequest) {

    //Request SPC
    keyRequest.makeStreamingContentKeyRequestData(forApp: appCertificate, contentIdentifier: assetID, options: keyRequestOptions) {
        (spcBytes: Data?, spcCreationError: Error?) in
        //Send SPC to Key Server and obtain CKC
        if let ckcBytes = getCKCFromKeyServer(forSPC: spcBytes) {
            //Create Response using the CKC you obtained from Key Server
            let response = AVContentKeyResponse(fairPlayStreamingKeyResponseData: ckcBytes)
            //Set Response on the Key Request object when you are about to start playback
            keyRequest.processContentKeyResponse(response)
        }
    }
}

Delegate callback that delivers AVContentKeyRequest

```swift
func contentKeySession(_ session: AVContentKeySession, didProvide keyRequest: AVContentKeyRequest) {
    // Request SPC
    keyRequest.makeStreamingContentKeyRequestData(forApp: appCertificate, contentIdentifier: assetID, options: keyRequestOptions) {
        (spcBytes: Data?, spcCreationError: Error?) in
            // Send SPC to Key Server and obtain CKC
            if let ckcBytes = getCKCFromKeyServer(forSPC: spcBytes) {
                // Create Response using the CKC you obtained from Key Server
                let response = AVContentKeyResponse(fairPlayStreamingKeyResponseData: ckcBytes)
                // Set Response on the Key Request object when you are about to start playback
                keyRequest.processContentKeyResponse(response)
            }
    }
}
```
func contentKeySession(_ session: AVContentKeySession, didProvide keyRequest: AVContentKeyRequest) {
    // Request SPC
    keyRequest.makeStreamingContentKeyRequestData(forApp: appCertificate, contentIdentifier: assetID, options: keyRequestOptions) {
        (spcBytes: Data?, spcCreationError: Error?) in
        // Send SPC to Key Server and obtain CKC
        if let ckcBytes = getCKCFromKeyServer(forSPC: spcBytes) {
            // Create Response using the CKC you obtained from Key Server
            let response = AVContentKeyResponse(fairPlayStreamingKeyResponseData: ckcBytes)
            // Set Response on the Key Request object when you are about to start playback
            keyRequest.processContentKeyResponse(response)
        }
    }
}
// Delegate callback that delivers AVContentKeyRequest
func contentKeySession(_ session: AVContentKeySession, didProvide keyRequest: AVContentKeyRequest) {
    // Request SPC
    keyRequest.makeStreamingContentKeyRequestData(forApp: appCertificate, contentIdentifier: assetID, options: keyRequestOptions) {
        (spcBytes: Data?, spcCreationError: Error?) in
        // Send SPC to Key Server and obtain CKC
        if let ckcBytes = getCKCFromKeyServer(forSPC: spcBytes) {
            // Create Response using the CKC you obtained from Key Server
            let response = AVContentKeyResponse(fairPlayStreamingKeyResponseData: ckcBytes)
            // Set Response on the Key Request object when you are about to start playback
            keyRequest.processContentKeyResponse(response)
        }
    }
}
Responding to Key Requests
Responding to Key Requests

Be mindful while responding to key requests
Responding to Key Requests

Be mindful while responding to key requests

• Secure decrypt slots - limited resource
Responding to Key Requests

Be mindful while responding to key requests

• Secure decrypt slots - limited resource
• Set CKC as response only for required key requests
Responding to Key Requests

Be mindful while responding to key requests

• Secure decrypt slots - limited resource
• Set CKC as response only for required key requests
• Respond to key requests just before requesting playback
Persistent FPS Keys
Persistent FPS Keys

Persistent FPS keys protect offline HLS assets
Persistent FPS Keys

Persistent FPS keys protect offline HLS assets

Use `AVContentKeySession` to

- Create persistent keys before requesting HLS asset download
Persistent FPS Keys

Persistent FPS keys protect offline HLS assets

Use `AVContentKeySession` to

- Create persistent keys before requesting HLS asset download

Simpler and cleaner workflow
Persistent FPS Keys

Persistent FPS keys protect offline HLS assets

Use `AVContentKeySession` to

- Create persistent keys before requesting HLS asset download

Simpler and cleaner workflow

Work with `AVPersistableContentKeyRequest`
//Delegate callback that delivers AVContentKeyRequest

func contentKeySession(_ session: AVContentKeySession, didProvide keyRequest: AVContentKeyRequest) {
    // Check if you are creating a Persistent Decryption Key
    if (creatingPersistentDecryptionKey(keyRequest.identifier)) {
        // Request a Persistable Key Request
        keyRequest.respondByRequestingPersistableContentKeyRequest()
        return
    }
    // Continue with AVContentKeyRequest while loading non-Persistent Decryption Key
}
//Delegate callback that delivers AVContentKeyRequest
func contentKeySession(_ session: AVContentKeySession, didProvide keyRequest: AVContentKeyRequest) {
    //Check if you are creating a Persistent Decryption Key
    if (creatingPersistentDecryptionKey(keyRequest.identifier)) {
        //Request a Persistable Key Request
        keyRequest.respondByRequestingPersistableContentKeyRequest()
        return
    }
    //Continue with AVContentKeyRequest while loading non-Persistent Decryption Key
}
func contentKeySession(_ session: AVContentKeySession, didProvide keyRequest: AVContentKeyRequest) {
    // Check if you are creating a Persistent Decryption Key
    if (creatingPersistentDecryptionKey(keyRequest.identifier)) {
        // Request a Persistable Key Request
        keyRequest.respondByRequestingPersistableContentKeyRequest()
        return
    }  
    // Continue with AVContentKeyRequest while loading non-Persistent Decryption Key
}
/Delegate callback that delivers AVContentKeyRequest

```swift
func contentKeySession(_ session: AVContentKeySession, didProvide keyRequest: AVContentKeyRequest) {
    // Check if you are creating a Persistent Decryption Key
    if (creatingPersistentDecryptionKey(keyRequest.identifier)) {
        // Request a Persistable Key Request
        keyRequest/respondByRequestingPersistableContentKeyRequest()
        return
    }
    // Continue with AVContentKeyRequest while loading non-Persistent Decryption Key
}
```
//Delegate callback that delivers AVContentKeyRequest
func contentKeySession(_ session: AVContentKeySession, didProvide keyRequest: AVContentKeyRequest) {
    //Check if you are creating a Persistent Decryption Key
    if (creatingPersistentDecryptionKey(keyRequest.identifier)) {
        //Request a Persistable Key Request
        keyRequest.respondByRequestingPersistableContentKeyRequest()
        return
    }
    //Continue with AVContentKeyRequest while loading non-Persistent Decryption Key
}

//Delegate callback that delivers AVPersistableContentKeyRequest
func contentKeySession(_ session: AVContentKeySession, didProvide keyRequest: AVPersistableContentKeyRequest) {
}

//Delegate callback that delivers AVPersistableContentKeyRequest

func contentKeySession(_ session: AVContentKeySession, didProvide keyRequest: AVPersistableContentKeyRequest) {
}

//Delegate callback that delivers AVPersistableContentKeyRequest
func contentKeySession(_ session: AVContentKeySession, didProvide keyRequest: AVPersistableContentKeyRequest) {

    //Request SPC
    keyRequest.makeStreamingContentKeyRequestData(forApp: appCertificate, contentIdentifier: assetID, options: keyRequestOptions) {
        (spcBytes: Data?, spcCreationError: Error?) in
        //Send SPC to Key Server and obtain CKC
        if let ckcBytes = getCKCFromKeyServer(forSPC: spcBytes) {
            //Request Persistent Decryption Key by providing the CKC
            let persistentKey = try? keyRequest.persistableContentKey(fromKeyVendorResponse: ckcBytes!, options: nil)

            //Stow persistentKey data blob for future use
        }
    }
}
// Delegate callback that delivers AVPersistableContentKeyRequest
func contentKeySession(_ session: AVContentKeySession, didProvide keyRequest: AVPersistableContentKeyRequest) {
    // Request SPC
    keyRequest.makeStreamingContentKeyRequestData(forApp: appCertificate, contentIdentifier: assetID, options: keyRequestOptions) { (spcBytes: Data?, spcCreationError: Error?) in
        // Send SPC to Key Server and obtain CKC
        if let ckcBytes = getCKCFromKeyServer(forSPC: spcBytes) {
            // Request Persistent Decryption Key by providing the CKC
            let persistentKey = try? keyRequest.persistableContentKey(fromKeyVendorResponse: ckcBytes!, options: nil)
        }
        // Stow persistentKey data blob for future use
    }
}
/Delegate callback that delivers AVPersistableContentKeyRequest

func contentKeySession(_ session: AVContentKeySession, didProvide keyRequest: AVPersistableContentKeyRequest) {
    //Request SPC
    keyRequest.makeStreamingContentKeyRequestData(forApp: appCertificate, contentIdentifier: assetID, options: keyRequestOptions) {
        (spcBytes: Data?, spcCreationError: Error?) in
        //Send SPC to Key Server and obtain CKC
        if let ckcBytes = getCKCFromKeyServer(forSPC: spcBytes) {
            //Request Persistent Decryption Key by providing the CKC
            let persistentKey = try? keyRequest.persistableContentKey(fromKeyVendorResponse: ckcBytes!, options: nil)

            //Stow persistentKey data blob for future use
        }
    }
}
/Delegate callback that delivers AVPersistableContentKeyRequest

```swift
func contentKeySession(_ session: AVContentKeySession, didProvide keyRequest: AVPersistableContentKeyRequest) {
    //Request SPC
    keyRequest.makeStreamingContentKeyRequestData(forApp: appCertificate, contentIdentifier: assetID, options: keyRequestOptions) {
        (spcBytes: Data?, spcCreationError: Error?) in
        //Send SPC to Key Server and obtain CKC
        if let ckcBytes = getCKCFromKeyServer(forSPC: spcBytes) {
            //Request Persistent Decryption Key by providing the CKC
            let persistentKey = try? keyRequest.persistableContentKey(fromKeyVendorResponse: ckcBytes!, options: nil)
        }
    }
    //Stow persistentKey data blob for future use
}
```
//Delegate callback that delivers AVPersistableContentKeyRequest

func contentKeySession(_ session: AVContentKeySession, didProvide keyRequest: AVPersistableContentKeyRequest) {
    //Create Response using the Persistent Key that was created earlier
    let response = AVContentKeyResponse(fairPlayStreamingKeyResponseData: persistentKey)
    //Set Response on the Persistable Key Request Object
    keyRequest.processContentKeyResponse(response)
}
// Delegate callback that delivers AVPersistableContentKeyRequest

func contentKeySession(_ session: AVContentKeySession, didProvide keyRequest: AVPersistableContentKeyRequest) {
    // Create Response using the Persistent Key that was created earlier
    let response = AVContentKeyResponse(fairPlayStreamingKeyResponseData: persistentKey)
    // Set Response on the Persistable Key Request Object
    keyRequest.processContentKeyResponse(response)
}
func contentKeySession(_ session: AVContentKeySession, didProvide keyRequest: AVPersistableContentKeyRequest) {
    // Create Response using the Persistent Key that was created earlier
    let response = AVContentKeyResponse(fairPlayStreamingKeyResponseData: persistentKey)

    // Set Response on the Persistable Key Request Object
    keyRequest.processContentKeyResponse(response)
}
AVContentKeySession
AVContentKeySession

Works similarly to AVAssetResourceLoader
AVContentKeySession

Works similarly to AVAssetResourceLoader

AVAssetResourceLoadingRequest → AVContentKeyRequest
AVContentKeySession

Works similarly to AVAssetResourceLoader

 AVAssetResourceLoadingRequest → AVContentKeyRequest

 AVAssetResourceLoaderDelegate → AVContentKeySessionDelegate
AVContentKeySession

Works similarly to AVAssetResourceLoader

AVAssetResourceLoadingRequest → AVContentKeyRequest

AVAssetResourceLoaderDelegate → AVContentKeySessionDelegate

Except not tied to AVURLAsset at the time of creation
AVContentKeySession

Works similarly to AVAssetResourceLoader

AVAssetResourceLoadingRequest → AVContentKeyRequest

AVAssetResourceLoaderDelegate → AVContentKeySessionDelegate

Except not tied to AVURLAsset at the time of creation

Create AVContentKeySession at any time
AVContentKeySession

Works similarly to `AVAssetResourceLoader`

- `AVAssetResourceLoadingRequest` → `AVContentKeyRequest`
- `AVAssetResourceLoaderDelegate` → `AVContentKeySessionDelegate`

Except not tied to `AVURLAsset` at the time of creation

Create `AVContentKeySession` at any time

Add asset as a content key recipient before requesting playback

- Using `addContentKeyRecipient()`
Recommended Usage
Recommended Usage

AVContentKeySession for

- Content decryption keys
Recommended Usage

**AVContentKeySession** for
- Content decryption keys

**AVAssetResourceLoader** for
- Playlists, media data, and metadata
Recommended Usage

AVContentKeySession for
• Content decryption keys

AVAssetResourceLoader for
• Playlists, media data, and metadata
• Key loading is still supported
Recommended Usage

**AVContentKeySession** for
- Content decryption keys

**AVAssetResourceLoader** for
- Playlists, media data, and metadata
- Key loading is still supported

What if **AVURLAsset** has both delegates installed?
Recommended Usage

**AVContentKeySession**
- Content decryption keys

**AVAssetResourceLoader**
- Playlists, media data, and metadata
- Key loading is still supported

What if **AVURLAsset** has both delegates installed?
- **AVAssetResourceLoader** delegate should defer key requests to **AVContentKeySession**
func resourceLoader(_ resourceLoader: AVAssetResourceLoader, shouldWaitForLoadingOfRequestedResource loadingRequest: AVAssetResourceLoadingRequest) -> Bool {
    // Check if the resource loading request is for Content Decryption Key
    if (requestIsForContentDecryptionKey(request: loadingRequest)) {
        // Defer loading of Content Decryption Key to AVContentKeySession
        loadingRequest.contentInformationRequest?.contentType = AVStreamingKeyDeliveryContentKeyType
        loadingRequest.finishLoading()
        return true
    }

    // Continue loading all other resources (playlists, media data & metadata)
    return true
}
func resourceLoader(_ resourceLoader: AVAssetResourceLoader, shouldWaitForLoadingOfRequestedResource loadingRequest: AVAssetResourceLoadingRequest) -> Bool {
    // Check if the resource loading request is for Content Decryption Key
    if (requestIsForContentDecryptionKey(request: loadingRequest)) {
        // Defer loading of Content Decryption Key to AVContentKeySession
        loadingRequest.contentInformationRequest?.contentType = AVStreamingKeyDeliveryContentKeyType
        loadingRequest.finishLoading()
        return true
    }

    // Continue loading all other resources (playlists, media data & metadata)
    return true
}
func resourceLoader(_ resourceLoader: AVAssetResourceLoader, shouldWaitForLoadingOfRequestedResource loadingRequest: AVAssetResourceLoadingRequest) -> Bool {

    // Check if the resource loading request is for Content Decryption Key
    if (requestIsForContentDecryptionKey(request: loadingRequest)) {

        // Defer loading of Content Decryption Key to AVContentKeySession
        loadingRequest.contentInformationRequest?.contentType = AVStreamingKeyDeliveryContentKeyType
        loadingRequest.finishLoading()
        return true
    }

    // Continue loading all other resources (playlists, media data & metadata)
    return true
}
func resourceLoader(_ resourceLoader: AVAssetResourceLoader, shouldWaitForLoadingOfRequestedResource loadingRequest: AVAssetResourceLoadingRequest) -> Bool {
    // Check if the resource loading request is for Content Decryption Key
    if (requestIsForContentDecryptionKey(request: loadingRequest)) {
        // Defer loading of Content Decryption Key to AVContentKeySession
        loadingRequest.contentInformationRequest?.contentType = AVStreamingKeyDeliveryContentKeyType
        loadingRequest.finishLoading()
        return true
    }

    // Continue loading all other resources (playlists, media data & metadata)
    return true
}
func resourceLoader(_ resourceLoader: AVAssetResourceLoader, 
shouldWaitForLoadingOfRequestedResource loadingRequest: AVAssetResourceLoadingRequest) -> Bool
{
    //Check if the resource loading request is for Content Decryption Key
    if (requestIsForContentDecryptionKey(request: loadingRequest)) {
        //Defer loading of Content Decryption Key to AVContentKeySession
        loadingRequest.contentInformationRequest?.contentType = AVStreamingKeyDeliveryContentKeyType
        loadingRequest.finishLoading()
        return true
    }

    //Continue loading all other resources (playlists, media data & metadata)
    return true
}
Dual Expiry Windows for Persistent Keys
Dual Expiry Windows for Persistent Keys

Support dual expiry windows for persistent FPS keys
Dual Expiry Windows for Persistent Keys

Support dual expiry windows for persistent FPS keys

Dual expiry window model for rentals
Dual Expiry Windows for Persistent Keys

Support dual expiry windows for persistent FPS keys

Dual expiry window model for rentals

Works for both online and offline playback
Dual Expiry Windows for Persistent Keys

Support dual expiry windows for persistent FPS keys

Dual expiry window model for rentals

Works for both online and offline playback

Server opts in by sending suitable descriptor in CKC
Dual Expiry Windows for Persistent Keys

Support dual expiry windows for persistent FPS keys

Dual expiry window model for rentals

Works for both online and offline playback

Server opts in by sending suitable descriptor in CKC

• Storage expiry
Dual Expiry Windows for Persistent Keys

Support dual expiry windows for persistent FPS keys

Dual expiry window model for rentals

Works for both online and offline playback

Server opts in by sending suitable descriptor in CKC

• Storage expiry
• Playback expiry
Dual Expiry Windows for Persistent Keys
Dual Expiry Windows for Persistent Keys
User rents content
Dual Expiry Windows for Persistent Keys

Create Persistent Key with CKC

User rents content
Dual Expiry Windows for Persistent Keys

Create Persistent Key with CKC

K1 Expires in 30 Days
Dual Expiry Windows for Persistent Keys

K1 Expires in 30 Days
Dual Expiry Windows for Persistent Keys

User starts playback

K1 Expires in 30 Days
Dual Expiry Windows for Persistent Keys

K1 Expires in 30 Days

User starts playback

Use K1 to answer Key Request
Dual Expiry Windows for Persistent Keys

Use K1 to answer Key Request

Receive updated Persistent Key

K1 Expires in 30 Days
Dual Expiry Windows for Persistent Keys

K1 Expires in 30 Days

K2 Expires in 24 Hours

Receive updated Persistent Key
Dual Expiry Windows for Persistent Keys

K1 is Expired

K2 Expires in 24 Hours

Receive updated Persistent Key
Dual Expiry Windows for Persistent Keys

K1 is Expired

K1

K2 Expires in 24 Hours

K2

K1 is Expired
Dual Expiry Windows for Persistent Keys

Updated Persistent Key sent through delegate callback:

`didUpdatePersistableContentKey`

K1 is Expired

K2 Expires in 24 Hours
Summary
HEVC video and IMSC1 subtitles now available in HLS
HEVC video and IMSC1 subtitles now available in HLS

New `EXT-X-GAP` tag and metavariable support in m3u8
Summary

HEVC video and IMSC1 subtitles now available in HLS

New **EXT-X-GAP** tag and metavariable support in m3u8

Synchronized playback of live HLS streams
Summary

HEVC video and IMSC1 subtitles now available in HLS

New `EXT-X-GAP` tag and metavariable support in m3u8

Synchronized playback of live HLS streams

Better control over offline storage and aggregate asset downloads
Summary

HEVC video and IMSC1 subtitles now available in HLS

New `EXT-X-GAP` tag and metavariable support in m3u8

Synchronized playback of live HLS streams

Better control over offline storage and aggregate asset downloads

`AVContentKeySession` API to manage and deliver content keys
Summary

HEVC video and IMSC1 subtitles now available in HLS

New \texttt{EXT-X-GAP} tag and metavariable support in m3u8

Synchronized playback of live HLS streams

Better control over offline storage and aggregate asset downloads

\texttt{AVContentKeySession} API to manage and deliver content keys

Rental support for persistent FPS keys
More Information

https://developer.apple.com/wwdc17/504
<table>
<thead>
<tr>
<th>Session</th>
<th>Location</th>
<th>Date and Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error Handling Best Practices for HLS</td>
<td>WWDC 2017 Video</td>
<td></td>
</tr>
<tr>
<td>HLS Authoring Update</td>
<td>WWDC 2017 Video</td>
<td></td>
</tr>
<tr>
<td>Introducing HEIF and HEVC</td>
<td>Executive Ballroom</td>
<td>Tuesday 4:10PM</td>
</tr>
<tr>
<td>Media and Gaming Accessibility</td>
<td>Grand Ballroom A</td>
<td>Wednesday 3:10PM</td>
</tr>
<tr>
<td>Introducing AirPlay 2</td>
<td>Executive Ballroom</td>
<td>Thursday 4:10PM</td>
</tr>
<tr>
<td>Working with HEIF and HEVC</td>
<td>Hall 2</td>
<td>Friday 11:00PM</td>
</tr>
<tr>
<td>Lab</td>
<td>Location</td>
<td>Time</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>----------------</td>
<td>------------------</td>
</tr>
<tr>
<td>AVFoundation Lab 1</td>
<td>Technology Lab F</td>
<td>Tue 1:00 - 4:10 PM</td>
</tr>
<tr>
<td>HTTP Live Streaming Lab 1</td>
<td>Technology Lab F</td>
<td>Wed 9:00 - 11:00 AM</td>
</tr>
<tr>
<td>AVFoundation Lab 2</td>
<td>Technology Lab G</td>
<td>Wed 11:00 AM - 1:00 PM</td>
</tr>
<tr>
<td>AVFoundation Lab 3</td>
<td>Technology Lab F</td>
<td>Thur 12:00 - 3:00 PM</td>
</tr>
<tr>
<td>HTTP Live Streaming Lab 2</td>
<td>Technology Lab G</td>
<td>Thur 3:10 - 6:00 PM</td>
</tr>
<tr>
<td>Lab Name</td>
<td>Location</td>
<td>Time</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------</td>
<td>---------------</td>
</tr>
<tr>
<td>AVKit Lab 1</td>
<td>Technology Lab F</td>
<td>Tue 1:00 - 4:10 PM</td>
</tr>
<tr>
<td>HEIF/HEVC Lab 1</td>
<td>Technology Lab A</td>
<td>Wed 9:00 - 11:00 AM</td>
</tr>
<tr>
<td>AirPlay Lab 1</td>
<td>Technology Lab A</td>
<td>Wed 11:00 AM - 1:00 PM</td>
</tr>
<tr>
<td>AVKit Lab 2</td>
<td>Technology Lab G</td>
<td>Wed 11:00 AM - 1:00 PM</td>
</tr>
<tr>
<td>AirPlay Lab 2</td>
<td>Technology Lab A</td>
<td>Fri 9:00 - 11:00 AM</td>
</tr>
<tr>
<td>HEIF/HEVC Lab 2</td>
<td>Technology Lab F</td>
<td>Fri 12:00 - 1:50 PM</td>
</tr>
</tbody>
</table>