HTTP Live Streaming (HLS) Explained

## **What is HTTP Live Streaming (HLS)?**

**HLS** (HTTP Live Streaming) is a widely adopted video streaming protocol used for both **on-demand** and **live streaming**. It works by breaking video into small, downloadable HTTP-based files that are played sequentially.

* Works on all internet-connected devices because it leverages standard HTTP.
* Supports **adaptive bitrate streaming**, adjusting video quality based on network conditions.
* Originally developed by Apple, it's now a ubiquitous standard across various devices and platforms.

## **What is Streaming?**

**Streaming** is a method of delivering audio and video content over the Internet continuously, allowing playback to begin before the entire file has been downloaded.

* The media can be stored remotely on a server or generated in real-time (for live streams).
* Enables a smooth viewing experience even while data is still being received.

## **What is HTTP?**

**HTTP** (Hypertext Transfer Protocol) is an application layer protocol used for transferring data between networked devices. While typically following a request-response pattern, in streaming, the connection often remains open, allowing the server to continuously push video data to the client.

## **How Does HLS Work?**

HLS involves a three-main-stage process: server-side preparation, distribution, and client-side playback.

### **On the Server**

1. **Encoding**: The original video is converted into a standard, web-friendly format, typically **H.264** or **H.265**.
2. **Segmenting**: The encoded video is then split into short, manageable segments (often 6-10 seconds by default).
3. **Index File Creation**: An index (or manifest) file is created, which lists the order of these segments for proper playback.
4. **Multiple Quality Versions**: To enable adaptive bitrate streaming, multiple versions of the video are produced at different quality levels (e.g., 480p, 720p, 1080p).

### **Distribution**

* The video segments and index files are delivered via standard HTTP.
* Often, a **Content Delivery Network (CDN)** is used to distribute these files globally, ensuring faster and more reliable delivery to users regardless of their location.

### **On the Client Device**

* The client player downloads the index file first to understand the video structure.
* It then uses this index to download and assemble the video segments in the correct order.
* Crucially, the client dynamically switches between the multiple quality levels provided, optimizing the viewing experience based on the user's current network speed.

## **Adaptive Bitrate Streaming in HLS**

This is a core feature of HLS. It allows the video player to **dynamically adjust the video quality** during playback based on the user's available network bandwidth.

* When bandwidth drops, the player switches to a lower quality (smaller file size) segment to prevent buffering.
* When bandwidth improves, it automatically switches back to a higher quality segment for a better visual experience.
* This is made possible by the server providing multiple pre-encoded quality versions of the video.

## **Does HLS Use TCP or UDP?**

**HLS uses TCP (Transmission Control Protocol), not UDP (User Datagram Protocol).**

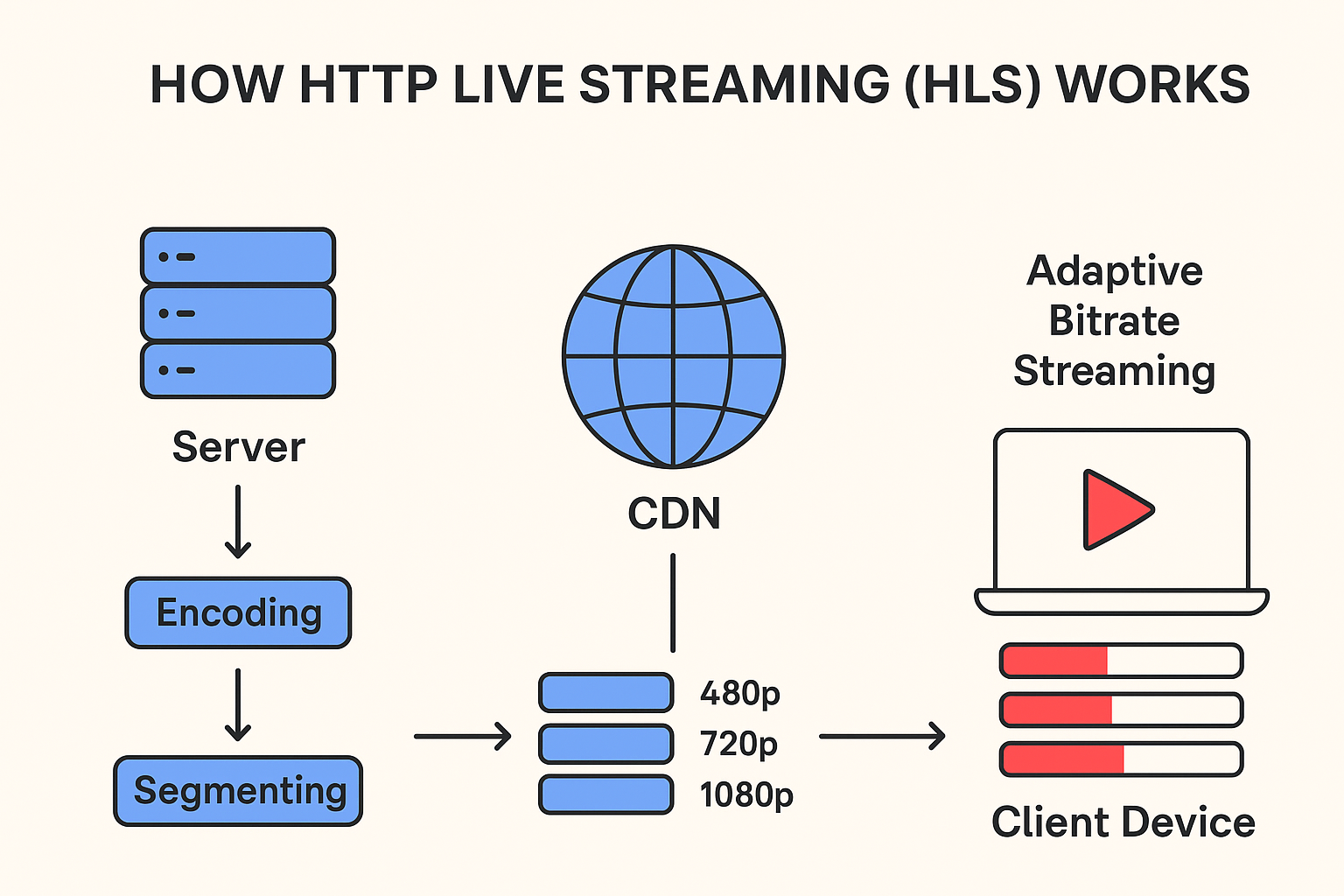
Here's why:

* **HTTP is built for TCP**: Since HLS relies on HTTP, it inherits TCP as its underlying transport protocol.
* **Reliable Connections**: Modern internet connections are generally reliable, and TCP ensures data packets are delivered in order and retransmits lost packets, which is crucial for continuous video playback.
* **Adaptive Bitrate Mitigation**: While TCP can be slower due to its reliability features, adaptive bitrate streaming helps mitigate potential issues by reducing the video quality when needed.
* **No Real-time Speed Requirement**: Unlike applications like video calls (which might use UDP for low-latency, real-time communication), streaming typically prioritizes consistent delivery over ultra-low latency.

## **Other Common Streaming Protocols**

While HLS is dominant, other protocols exist:

* **MPEG-DASH**: Similar to HLS, it's an HTTP-based adaptive bitrate streaming protocol, widely used in non-Apple ecosystems.
* **HDS (HTTP Dynamic Streaming)**: Another HTTP-based protocol, primarily associated with Adobe Flash (now largely deprecated).
* **RTMP (Real-Time Messaging Protocol)**: Still in use for ingesting live streams to platforms, but its use for client-side playback has declined significantly since Flash support ended.



## Summary Table of AWS HLS Support

| **AWS Service / Solution** | **Role in HLS Workflows** |
| --- | --- |
| MediaLive | Encodes live streams into ABR HLS |
| MediaStore | Stores segmented HLS content |
| MediaPackage | Packages and delivers HLS / LL-HLS (with features) |
| Amazon Kinesis Video Streams | Provides HLS playback for live and archived video |
| CloudFront | Global delivery and acceleration for HLS |
| CloudFormation Live Solutions | Rapid deployment of complete HLS infrastructure |

This document outlines secure and efficient methods for storing, delivering, and displaying course videos, with a focus on preventing unauthorized downloads.

## 1. Where to Store Course Videos

### Option A — AWS Media Services (Best for Professional-Grade Streaming)

This option provides robust control and scalability, ideal for professional-grade streaming.

* **Storage:**
  + Store raw course videos in **Amazon S3** (private bucket).
  + Use **AWS Elemental MediaConvert** to encode videos into HLS or CMAF format (adaptive bitrate formats).
* **Delivery:**
  + **AWS Elemental MediaPackage** (or **CloudFront** as CDN) delivers adaptive bitrate streaming.
  + Configure **signed URLs** (CloudFront Signed Cookies/URLs) to ensure only logged-in users can stream.
* **Security:**
  + Enable **DRM** (Widevine/FairPlay/PlayReady) via AWS MediaPackage for playback encryption.
  + Implement **tokenized playback** where your backend signs requests, ensuring only enrolled users can access.

### Option B — Specialized Video Hosting (Faster Setup, Less Maintenance)

For quicker setup and reduced operational overhead, consider specialized video hosting platforms.

Platforms like **Vimeo OTT, Mux, Cloudflare Stream, or Wistia** handle:

* Storage
* Adaptive bitrate streaming
* DRM/encryption
* Signed URLs

These platforms typically offer APIs for integration with your website.

* **Pros:** Quick to set up, requires less AWS specific knowledge.
* **Cons:** Involves monthly costs per GB or per viewer.

## 2. How to Display the Video on Your Site

To display videos, you'll need a **course player** with specific capabilities:

* Supports **HLS streaming** (m3u8 playlists).
* Integrates with your **authentication system** to verify user enrollment before serving video content.

**Examples of video players:**

* **Video.js** with HLS plugin
* **Shaka Player**
* **JW Player** (a paid option with built-in DRM support)

## 3. How to Prevent Downloads

While no method is 100% foolproof (screen recording is always a possibility), you can implement several measures to make direct downloading extremely difficult.

### Technical Protections:

* **HLS with Encryption:**
  + Use **AES-128 or SAMPLE-AES encryption** for video segments.
  + Encryption keys are delivered only to authenticated players via HTTPS.
* **Signed URLs / Tokens:**
  + Each video request must include a **short-lived signed token** generated by your backend.
* **Disable Right Click & Direct File Links:**
  + Ensure video files are not publicly accessible, preventing users from simply using "Save As".
* **DRM (Digital Rights Management) — Strongest Protection:**
  + Utilizes standards like **Widevine** (Chrome/Android), **FairPlay** (Safari/iOS), and **PlayReady** (Edge/Windows).
  + **DRM prevents direct access to decrypted video segments**, offering the highest level of content protection.

## 4. Suggested Architecture for Your Use Case

For a custom Learning Management System (LMS) or course site requiring secure video streaming, the following architecture is recommended:

1. **Upload Videos** → **Amazon S3** (private bucket)
2. **Transcode** → **AWS MediaConvert** to HLS formats (supporting multiple qualities for adaptive streaming)
3. **Store Metadata** → Course information and enrolled users managed in your **database** (e.g., MySQL/Postgres)
4. **Distribute** → **AWS CloudFront** with signed URLs for content delivery
5. **Player Integration** → Embed **Video.js** or **Shaka Player** within your course pages
6. **Access Control** → Your backend verifies user enrollment and then generates the necessary **signed playback URLs** for the video player.

| **Purpose** | **AWS Service** | **Notes** |
| --- | --- | --- |
| Store original videos | **Amazon S3 (private bucket)** | Never make the bucket public |
| Transcode into streaming format | **AWS Elemental MediaConvert** | Output as HLS (.m3u8 + .ts/.mp4 segments) |
| Package & encrypt video | **AWS Elemental MediaPackage** | Adaptive bitrate + DRM |
| Distribute securely | **Amazon CloudFront** | Signed URLs / signed cookies |
| Authentication & control | Your backend API | Checks enrollment before issuing video access |

## 2. Step-by-Step Implementation

This section provides a detailed step-by-step guide for implementing secure video streaming using AWS services.

Step 1 — Create Private S3 Bucket for Videos

* Go to AWS S3 → Create bucket (e.g., mysite-course-videos).
* Block all public access to this bucket.
* Upload your raw video files (e.g., .mp4) into this bucket.
* Set a bucket policy to ensure only MediaConvert and your backend IAM role can access it.

Step 2 — Convert Videos to HLS

* Open AWS MediaConvert.
* Create a new job:
  + Input: Specify your raw video from the S3 bucket created in Step 1.
  + Output group: Select Apple HLS.
  + Output qualities: Define multiple adaptive bitrate outputs, for example, 1080p, 720p, 480p.
  + Output destination: Specify another private S3 bucket (e.g., mysite-hls-output) where the transcoded outputs will be stored.
* Run the job. Upon completion, you will receive:
  + A .m3u8 master playlist file.
  + Multiple .ts segment files for each quality.

Step 3 — Package and Encrypt with MediaPackage

* Create a Channel in AWS Elemental MediaPackage.
* Create an Endpoint for this channel:
  + Type: HLS.
  + Enable encryption (e.g., AES-128 or DRM technologies like Widevine/FairPlay/PlayReady).
  + Connect this endpoint to your HLS output from MediaConvert (the .m3u8 playlist).
* Test playback using the AWS MediaPackage preview feature to ensure correct setup.

Step 4 — Distribute Securely with CloudFront

* Create a CloudFront distribution, using the MediaPackage endpoint as the origin.
* Enable Signed URLs or Signed Cookies:
  + This ensures that only users with a valid signed token from your backend can watch the video.
  + These tokens should expire quickly (e.g., 60 seconds) to prevent sharing.
* Restrict access to your MediaPackage origin so that only CloudFront can pull content from it.

Step 5 — Backend Enrollment Check

Your backend will handle the access control logic:

* Logic:
  + When a user clicks on a course, the frontend sends a request to your backend (e.g., /getVideoLink?courseId=123).
  + The backend then checks if the userId is enrolled in the courseId.
  + If the user is enrolled, the backend generates a CloudFront signed URL for the HLS .m3u8 master playlist.
  + The backend sends this signed URL back to the frontend.
* The frontend video player then uses this signed URL to play the video content.

Example: Node.js Signed URL Generation  
const AWS = require('aws-sdk');

const cloudfront = new AWS.CloudFront.Signer(process.env.CF\_KEY\_PAIR\_ID, process.env.CF\_PRIVATE\_KEY);

function getSignedUrl(fileUrl) {

const expires = Math.floor((Date.now() + 60 \* 1000) / 1000); // URL expires in 1 minute

return cloudfront.getSignedUrl({

url: fileUrl,

expires

});

}

Step 6 — Video Player Integration

Integrate a video player like Video.js or Shaka Player into your course page:

<video id="coursePlayer" class="video-js" controls preload="auto" width="800" height="450">

<source src="{{SIGNED\_HLS\_URL}}" type="application/x-mpegURL">

</video>

* This setup supports adaptive bitrate streaming, providing an optimal viewing experience based on the user's network conditions.
* It works consistently across various browsers and devices.

Step 7 — Prevent Downloads

To minimize the possibility of unauthorized downloads:

* Do NOT expose direct .ts or .mp4 URLs; always serve content via signed CloudFront URLs.
* Utilize AES-128 / DRM to encrypt the video streams.
* Short-lived tokens make any shared URLs useless after their expiration.
* While not foolproof, you can implement minor deterrents like disabling right-click context menus and hints in developer tools on the video player.
* For the highest level of security, implement Widevine/FairPlay DRM.

### Workflow Diagram

This diagram illustrates the flow of video content from storage to delivery:

S3 (Raw Videos) ↓ MediaConvert (Transcode to HLS) ↓ MediaPackage (Package + Encrypt) ↓ CloudFront (Signed URLs/CDN) ↓ Your Website (Video Player) ↓ Enrolled Users Only (Backend check before URL issue)

Workflow Diagram

S3 (Raw Videos)

↓

MediaConvert (Transcode to HLS)

↓

MediaPackage (Package + Encrypt)

↓

CloudFront (Signed URLs/CDN)

↓

Your Website (Video Player)

↓

Enrolled Users Only (Backend check before URL issue)

