License Servers & Key Delivery — Quick Reference

How a player gets the key to decrypt — and where every secret must stay. Encrypted video is useless without the key; the whole job is to deliver it to the right device for one playback and nowhere else. Engineering guidance; confirm vendor and device support live, it changes.

1 · THE LICENSE-ACQUISITION FLOW (EME)

- ☐ **encrypted -> generateRequest()** — the player opens a key session and asks the CDM to build a request.
- ☐ **message** — the CDM emits a device-signed license request; your app (JS) transports it, never reads it.
- ☐ **POST + entitlement token** — the app forwards the request to the license server with a short-lived token.
- ☐ **update()** — the server returns the key wrapped for the device; the key lands inside the CDM. Video plays.

2 · THE SECRET MAP — WHO NEVER SEES THE KEY

- ☐ **Content key (CEK)** is clear ONLY in: KMS/HSM, the license server, and the device CDM/TEE. Nowhere else.
- ☐ **default_KID** is the public NAME of the key — safe in the manifest. Never confuse the name with the key.
- ☐ **Communication key** (entitlement service ↔ license server) NEVER ships to the client — leak = minted tokens.
- ☐ **Never** put the clear key in the manifest, the CDN, the player JS, or the logs.

3 · THE THREE DIALECTS

- ☐ **Widevine** — CDM sends a device-signed **license request**; server returns a license. Android, Chrome, TVs.
- ☐ **PlayReady** — client sends a **license challenge** (XML) + KID; server returns a license. Windows, Xbox, TVs.
- ☐ **FairPlay** — device sends an **SPC**; the KSM checks the cert and returns a **CKC** with the key. Apple only.
- ☐ **Multi-DRM** speaks all three from one cbcs package — one workflow, every device.

4 · SESSION TYPES & LIVE ROTATION

- ☐ **temporary** — license dies with the tab; default for online. No secret on disk.
- ☐ **persistent-license** — saved to disk for OFFLINE playback; a deliberate secret on disk, apply rules strictly.
- ☐ **Key rotation (live)** — rotate the content key by interval; deliver keys hierarchically (batched) or the spike DDoSes your license server.

WHY A LICENSE PROXY — THE LAST DOOR TO THE KEY

The license server knows how to issue a key; deciding WHO gets one is your entitlement service, and it must live on the server. The clean pattern is a license proxy: the player sends its request with no secret; the proxy checks subscription, region, and concurrent-stream limit, mints a fresh single-use token (seconds, not hours), and forwards the request to the real license server. Putting the rights check in the JavaScript player is bypassed by anyone who opens developer tools; the communication key in the client lets an adversary mint valid tokens on your behalf. Keep entitlement on the server, keep tokens short and single-use, and the player just sees an endpoint that always answers. Confirm current vendor support; it changes.