Oriol Sánchez 253207
Mar Climente 252305

CDN & Docker for encoding

2.-)Now that you have your container ready, download any video you like from the internet (if lack of ideas, try BBB). Cut it to 1 minute and try to package it like this:

1. **MP4 container with HLS**
   Video .h264 AVC, audio AAC

   We convert to HLS using the following ffmpeg cmd:

```
ffmpeg_cmd = [
    "ffmpeg",
    "-i", video_path,
    "-c:v", "libx264",
    "-c:a", "aac",
    "-hls_time", "10",
    "-hls_playlist_type", "vod",
    "-f", "hls",
    output_hls_path
]
```

2. **MKV container with MPEG-DASH**
   Video VP9, audio AAC

   We convert to MPEG-DASH using the following ffmpeg cmd:

```
ffmpeg_cmd = [
    "ffmpeg",
    "-i", video_path,
    "-c:v", "libvpx-vp9",
    "-c:a", "aac",
    "-f", "dash",
    output_dash_path
]
```

Oriol Sánchez 253207
Mar Climente 252305

3.-) Now that you know how to 'Docker', search for the Bento4 software. Put it inside a Docker, and try to apply a DRM for the Previous packaged file. Some remarks: · If you're able to create an encoding ladder, that would be great! · Maybe you need to go to the mp4 file and package again before applying DRM (mp4encrypt)

First, we downloaded Bento4 from Bento4 Downloads for Linux since Docker runs on a Linux container. We can see that mp4encrypt is located inside .../Bin.

Next, we modified our Dockerfile to integrate Bento4 by adding the necessary configurations.

```
# Copy Bento4 binaries from your local machine into the container
COPY ./Bento4/bin /opt/bento4/bin

# Set environment variables so Bento4 can be used
ENV PATH="/opt/bento4/bin:${PATH}"
```

Then, we implemented several functions with corresponding endpoints to apply DRM protection.

```
def generate_encryption_keys():
    """Generate a random KID (Key ID) and encryption key"""
    kid = secrets.token_hex(16)
    key = secrets.token_hex(16)
    return kid, key
```

Oriol Sánchez **253207**
Mar Climente **252305**

```python
@app.post("/encrypt/")
async def encrypt_video(request: Request, file: UploadFile = File(...)):
    """
    Encrypts an uploaded fragmented MP4 file using Bento4's mp4encrypt.
    """
    # Generate unique filename
    file_ext = file.filename.split(".")[-1]
    video_filename = f"{uuid.uuid4()}.{file_ext}"
    video_path = os.path.join(UPLOAD_DIR, video_filename)

    # Save uploaded file
    with open(video_path, "wb") as buffer:
        buffer.write(await file.read())

    # Create output directory
    output_folder_name = video_filename.rsplit(".", 1)[0]
    output_folder = os.path.join(PROCESSED_DIR, output_folder_name)
    os.makedirs(output_folder, exist_ok=True)

    # Define encrypted output file path
    encrypted_file_path = os.path.join(output_folder, "encrypted.mp4")

    # Generate encryption keys
    kid, key = generate_encryption_keys()

    # Run Bento4 mp4encrypt command
    encrypt_cmd = [
        "mp4encrypt",
        "--method", "MPEG-CENC",
        "--key", f"1:{kid}:{key}",
        video_path,
        encrypted_file_path
    ]

    try:
        subprocess.run(encrypt_cmd, check=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
    except subprocess.CalledProcessError as e:
        return JSONResponse(
            status_code=500,
            content={"error": "Encryption failed", "details": e.stderr.decode()}
        )

    # Construct the URL for the encrypted video
    base_url = str(request.base_url)
    encrypted_url = f"{base_url}processed/{output_folder_name}/encrypted.mp4"

    return JSONResponse(content={"encrypted_video_url": encrypted_url, "kid": kid, "key": key})
```

1. **generate_encryption_keys()**
   This function generates a random encryption key and a Key ID (KID), both being 16-byte hex values. The KID serves as an identifier for the encryption key. These keys are essential for encrypting the video file using Bento4. The function returns both the key and KID, which will later be used during the encryption process.
2. **encrypt_video()**
   This function takes an uploaded MP4 file, whether for HLS or DASH, and saves it in the uploads/ folder. It then creates an output directory inside processed/ to store the encrypted version. Using Bento4's mp4encrypt, it encrypts the video with the previously generated key and outputs the encrypted file. Finally, it returns the URL where the encrypted video can be accessed.

However, we are encountering an issue when calling mp4encrypt to apply encryption:

Oriol Sánchez **253207**
Mar Climente **252305**

2025-02-23 00:43:55 FileNotFoundError: [Errno 2] No such file or directory: 'mp4encrypt'

Applying DRM to HLS Content

To protect HLS content, we encrypt each HLS segment using AES-128 encryption. The process starts by generating a random 128-bit AES key and creating a key info file. This file consists of three lines:

1. The URL where the key is or will be served.
2. The local file path where FFmpeg will save the key.
3. The AES key in hexadecimal format.

The key info file is crucial, as it instructs FFmpeg on how to encrypt the video segments.

We then execute FFmpeg with the -hls_key_info_file option, providing the generated key info file. FFmpeg processes the video, encrypts the segments, and outputs an HLS stream protected by DRM.

At least, we want to apply DRM to one container within the HLS stream. However, this method does not provide full DRM support like Bento4, Widevine or FairPlay, which require encryption with a DRM license server. Instead, it offers basic encryption at the segment level, restricting unauthorized access.

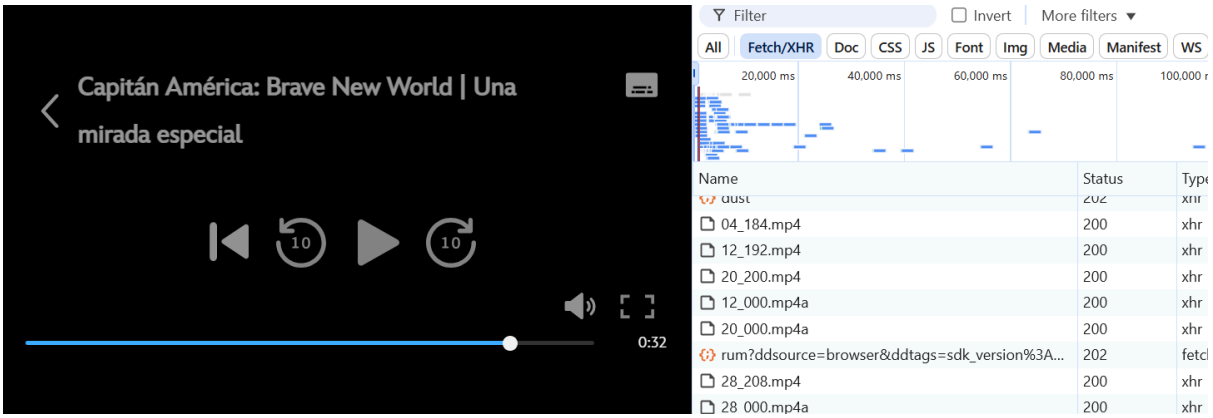
You can look at the result doing the dockerization and trying our functions. Also inside; GitHub\P4_Equips_Video\P4\app\static, you can find the 2 packages.

Oriol Sánchez **253207**
Mar Climente **252305**

4.-) Now that you understand video streaming, let's do something:
Connect to your favourite VOD platform · Use the developer tools from your browser · Investigate about what technology you're using: · HLS/DASH? · Codec? · DRM? Do a small report about it

Let's analyze Disney +.

Using the developer tools (Google Chrome), we can see the following:



By using Google Chrome's Developer Tools, we observed that Disney Plus streams video and audio using **.mp4** and **.mp4a** files. These files are delivered via XHR and Fetch requests, indicating an HTTP-based adaptive streaming method where content is split into smaller fragments. Although we did not find explicit HLS or DASH manifests (**.m3u8** or **.mpd**), the segmented delivery strongly suggests an adaptive bitrate streaming approach, allowing the video quality to adjust dynamically based on network conditions.

The use of **.mp4** files implies that Disney Plus relies on widely used video codecs such as H.264 (AVC) or HEVC (H.265), both of which are common in MP4 containers for online streaming. These codecs provide efficient compression and high-quality video at various bitrates, making them ideal for adaptive streaming.

While we did not find direct evidence of DRM technologies like Widevine, PlayReady, or FairPlay in network requests, the protected URL structure and the nature of the content suggest that DRM is in place to secure video streams.

In conclusion, Disney Plus appears to use adaptive streaming with fragmented **.mp4** files, likely encoded in H.264 or HEVC, while employing DRM mechanisms to protect its content.