UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Telecomunicació
i Aeroespacial de Castelldefels

MASTER THESIS

TITLE: Common Media Application Format. Implementation and Analysis

MASTER DEGREE: Master's degree in Applied Telecommunications and Engineering Management (MASTEAM)

AUTHOR: Gerard Solé i Castellví

ADVISOR: Juan López Rubio

SUPERVISOR: Javier López Rubio

DATE: July 8, 2019

Títol: Common Media Application Format. Implementació i anàlisi
Autor: Gerard Solé i Castellví

Director: Juan López Rubio

Supervisor: Javier López Rubio

Data: 8 de juliol de 2019

Resum

Les empreses de streaming de vídeo sobre internet estan en auge, transmetent tota classe de continguts a un públic cada vegada més gran. En l'àmbit operacional, tot i que el streaming sobre HTTP està estandarditzat hi ha hagut molta fragmentació en el mercat a causa de la mancança de col·laboració entre empreses de streaming i dispositius. Aquesta fragmentació acaba degenerant en una ineficiència per a les empreses, tant des del punt de vista tecnològic com de l'operacional i producte. Per a solucionar això, grans empreses del sector del streaming com ara Amazon, Apple, Google, Microsoft i Netflix, junt amb altres empreses del món de l'electrònica com ara Samsung i LG, s'han posat d'acord per dissenyar un nou format de fitxer preparat per trencar amb la fragmentació existent. Aquest nou format s'anomena *Common Media Application Format* (CMAF) i s'espera que sigui àmpliament acceptat per totes les empreses relacionades amb el món del streaming.

En aquest document es presenta una prova pilot realitzada a RakutenTV estudiant la implementació de CMAF en les operacions de l'empresa i analitzant la viabilitat de la solució. Com a moltes altres empreses, Rakuten TV sofreix de la fragmentació en el sector del streaming el qual té un impacte directe en el rendiment de la plataforma i en el cost econòmic de mantenir-la. Amb la nova solució s'espera poder oferir un servei més eficient i reduir l'impacte econòmic derivat de la gestió i l'emmagatzematge dels fitxers actuals.

Title : Common Media Application Format. Implementation and Analysis
Author: Gerard Solé i Castellví

Advisor: Juan López Rubio

Supervisor: Javier López Rubio

Date: July 8, 2019

Overview

Internet streaming companies are an important business nowadays streaming all kinds of content to a growing audience. Although streaming over HTTP is standardized, there has been a lot of fragmentation in the market due to the lack of collaboration between streaming companies and devices, this affects the company operations because of how they manage all the technologies together. This fragmentation ends up degenerating into an inefficiency for the companies, from the technological, operational and productive point of view. To solve this issue, large companies in the streaming sector such as Amazon, Apple, Google, Microsoft and Netflix, along with other companies in the electronics world, like Samsung and LG, have agreed to design a new file format ready to break with the existing fragmentation. This new format is called *Common Media Application Format* (CMAF) and is expected to be widely accepted by all companies related to the world of streaming.

This document presents a test carried out at RakutenTV studying the implementation of CMAF in the operations of the company and analyzing the its feasibility for a long-term run. Like many other companies, RakutenTV suffers from fragmentation which has a direct impact on the performance of the platform and the economic cost of maintaining it. With the new solution, it is expected to be able to offer a more efficient service and reduce the economic impact derived from the management and storage of current files.

A tothom qui m'ha ensenyat, i a tothom que ha volgut apendre de mi.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

Online streaming services have become an important business for multiple kind of companies. With the introduction of platforms like Netflix, HBO and RakutenTV people is getting used to watch any content at the time they want. This creates an interesting market niche where companies can make profit. *Motion Picture Association of America* (MPAA) published a study in 2018 where the American home entertainment market was analyzed, Figure 1 shows the streaming service growth from 2014 to 2018. In the American market it has surpassed the PayTV (cable) in subscriptions, and the tendency is that it will continue growing with new types of content.

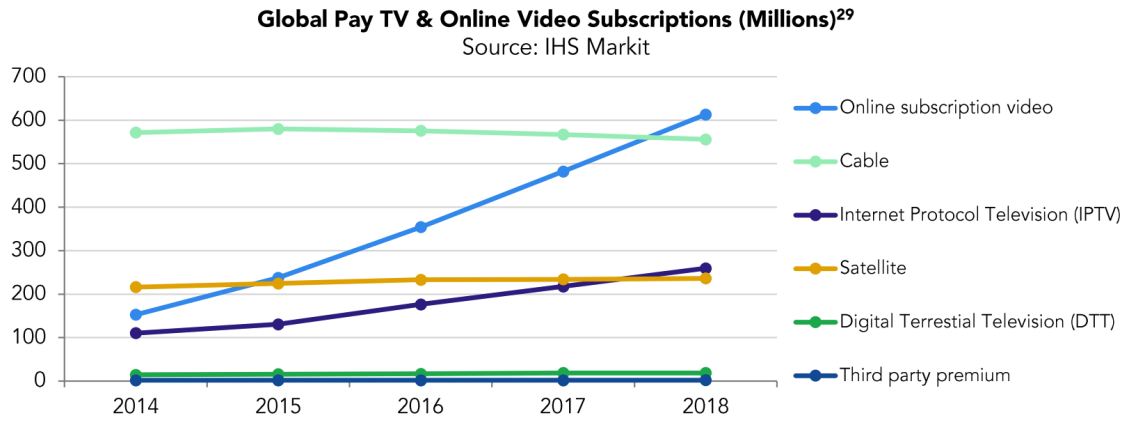

Figure 1: Year over year is streaming services are growing compared to old content broadcasting methods. Original image obtained from *Motion Picture Associate of America* [1].

RakutenTV is an European *Video-on-Demand* (VoD) company, it was founded back in 2009 under the name of *Wuaki.tv*, on 2012 it was bought by Rakuten. Its main focus is on transactional streaming (or Pay-per-View) where customers can buy or rent a movie in the platform. In 2019, the company is facing a continental expansion, it will be present in more than 42 countries. This thesis is related to the streaming technologies, seeking on how to make it efficient in a company like RakutenTV.

RakutentTV, as other companies in the streaming market, suffer from the fragmentation issues caused by the large number of devices to serve and which standards do they support. The biggest problem for streaming companies is how to manage all the packages from an efficient way. Streaming packages are basically the video streams stored in a certain format and they can be streamed using multiple kind of protocols, devices tend to support only some of them. If companies need to manage too many package formats, the storing and operational costs will grow and make the platform unprofitable.

To solve these issues, companies related to the streaming business, both service providers and device manufacturers have designed a new standard which defines how the streaming packages must be built and processed. Service providers like Apple, Microsoft and Google, and manufacturers such as LG and Samsung have been involved in the definition of the *Common Media Application Format* (CMAF). This new package takes profit from the knowledge acquired from the other streaming protocols, it defines how the videos must be processed and protected, and how to store the data in a well-known format so players can

play the content.

CMAF implementation and viability has been studied for RakutenTV, if it is suitable to be included in the company operations and how it could be economically feasible. In this thesis the implementation possibilities will be analyzed and validated against current market devices. Apart from this, some estimations on the package implementation inside the company will be done.

CHAPTER 1. PROJECT OVERVIEW

The idea behind this project comes from the Playback Department at RakutenTV. This company provides a streaming service of movies and shows at European level. The work carried out during this thesis is a proof-of-concept for future technical implementations inside the company, on the playback area.

In this chapter there will be an introduction to the streaming chain at Rakuten TV, followed by some technical introductions and finally which is the motivation for this thesis.

1.1. Playback chain at RakutenTV

Playback department in RakutenTV is in charge of the whole streaming chain developing technical solutions to improve the picture and streaming responsiveness of the platform. Figure 1.1 shows the whole playback chain in the company starting from the provider's master and ending in the device, where the playback session of the media is done. It is important to introduce how this process works to have a global vision of streaming on a company.

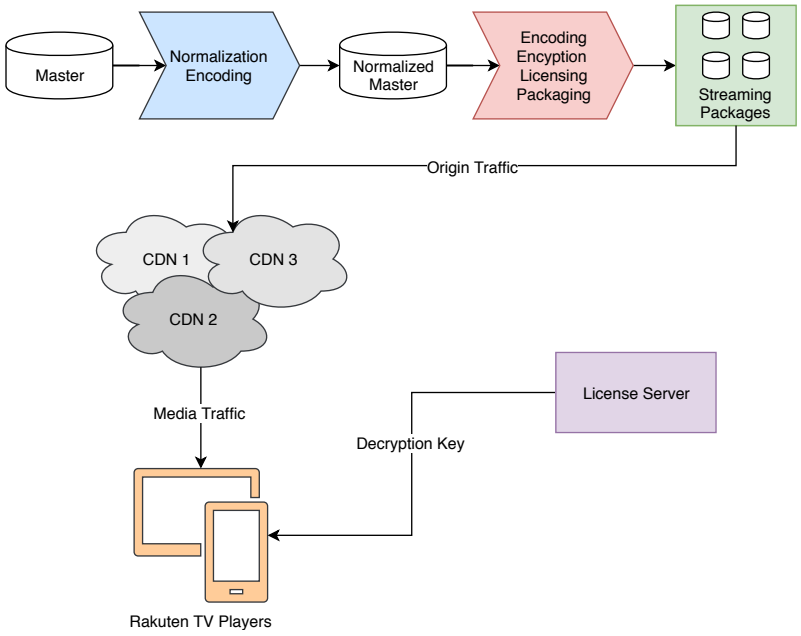

Figure 1.1: Playback chain at RakutenTV.

When the company acquires a master video from a cinema studio it can be delivered using multiple ways, there is a team inside the company in charge of obtaining such material and upload it to the ingestion platform. There are multiple ways to obtain the master, some companies push them to a remote file server, others provide websites where content can be downloaded, and finally there are others that send directly hard drives to the office. This introduces a complex start point, each material must be treated differently because of there is not a standardized way to supply the medias nor the medias are equal between them. Each master file can have different properties, some of them may use common

specifications and technologies, others not. Maybe some master files contain black bars on the top and bottom of the picture, but others use the whole display, etc. This forced the company to standardize a very accurate process to manipulate, generate and stream each media in the same way.

The following subsections introduce the four big steps that are carried out on the playback chain.

1.1.1. Normalization of the master

Imagine dealing with the material received from studios, those masters will not have a common format, if each step in the stream processing pipeline requires to support all the possible formats the software will not be maintainable. When writing the software to be used in each step of the pipeline it is better to have a common way to process the files, knowing which is the input format simplifies a lot the software development. A standard file will reduce the technical complexity and the possible failure points.

Because of this, the master normalization is the first step in the streaming chain. The normalization process will ingest any kind of material and process it so the output of the step is a well-known formatted video. This is known as the normalization encoding, and the output will be a normalized master that can be treated always in the same way on further steps. Within the normalization process some metadata information is added to the file, things such as the title name, provider, format of the video, and even the languages and subtitles are stored so next steps can process automatically the file without human intervention.

1.1.2. Package generation

Normalized masters cannot be streamed through network, they must be processed and prepared to do so. This is the step known as packaging, there is a second encoding of the normalized master where different videos are generated from the input file. These videos are known as renditions, and they are created to satisfy different conditions while streaming. Some renditions have a really great quality, while others have lower quality, but they can be streamed through the network without requiring so much bandwidth.

Film industry requires strict protection policies to avoid piracy on the VoD platforms. Because of this, the companies are required to protect the renditions using approved and secure methods certified by the studios and publishers. Usually this protection is achieved by using encryption algorithms together with secure key transmission between the VoD company and the players. Once the renditions are encoded and encrypted, the outcome of this step is a bunch encrypted video files. These encrypted video files are usually described on a manifest which holds the information of each rendition, their properties and the encryption method applied to the files. A player will use the manifest to fetch and play the content, within the manifest there is information on how to obtain the decryption key of the content.

This group of files (manifest and renditions) is a streaming package and it must be reachable through HTTP to enable the devices play the content. Figure 1.2 shows a schema of the contents of a streaming package.

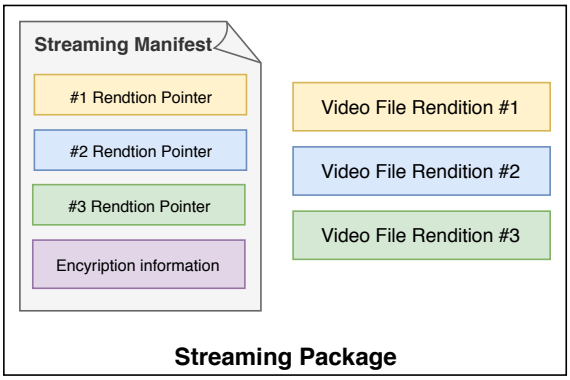

Figure 1.2: Streaming package representation. Contains a Manifest file with pointers to the video rendition files and the encryption information.

During this thesis, three streaming manifests will be presented, each one is used in a different streaming protocol. Just for future reference, they are Microsoft Smooth Streaming (MSS), Dynamic Adaptive Streaming over HTTP (DASH) and HTTP Live Streaming (HLS).

1.1.3. Package delivery through network

Once the streaming packages are built and stored, the delivery team will be in charge of making them available through internet. Streaming video is a resource demanding process, good network speeds and peering are required to satisfy all the playback sessions that may occur at a given moment. The company use several Content Delivery Networks (CDN) that can provide all the required bandwidth and content distribution. CDNs are perfect for cases such as streaming, they fetch the content from the original location and store it within their own network.

From a simplified point of view, a CDN can be seen as large distributed cache for content. When a CDN receives a content request the following steps will happen:

1. CDN checks the closest server (cache) to the user that is requesting the content.
2. The closest server will check if it already has the content cached, if it is there it will be returned directly to the customer.
3. If the content is not present on the server, the server itself will check on other servers within the CDN if they have the content. If the content is found it will be cached and then returned
4. Finally, if the requested content is not found withing the CDN, the server will fetch the content from the origin (where medias are always available), store it locally and send it back to the customer.

CDNs are distributed across regions, and they have multiple Points of Presence (PoP) where the caches are present. Having PoPs closer to the final customers will improve the throughput and latency between the player and the content.

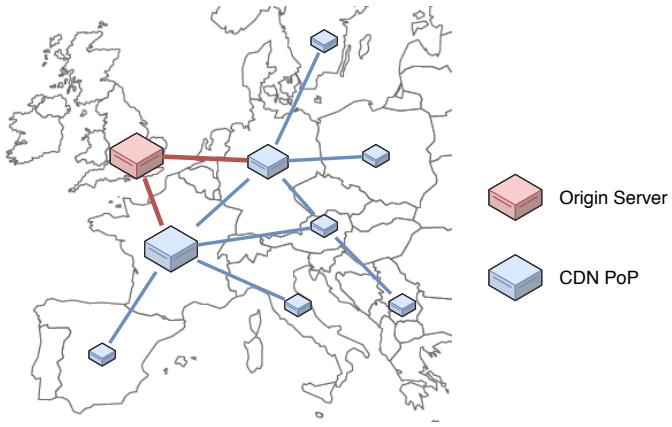

Figure 1.3: CDN Network representation with multiple Points of Presence and its connection to the Origin Server where packages are stored.

Figure 1.3 shows a simple representation of a CDN, the PoPs are distributed in different regions and are connected between them. They can reach the origin servers to fetch the content and spread it within the caches.

When partnering with CDNs it is important to check their network capacity and their support for the streaming protocols that are going to be used. Current streaming protocols work over HTTP making CDNs suitable for such delivery case.

1.1.4. Content key delivery

Streaming packages have been secured and encrypted, no one will be able to steal or expose the content on the clear in Internet, the players need the content key to decrypt the stream. Movie studios enforce the usage of Digital Rights Management (DRM) systems in VoD services. They ensure the security and confidentiality of the content key, DRM helps on keeping the content secure and encrypted while giving service to the customer. A customer has the rights to watch a movie, but not to hold and store the content in clear.

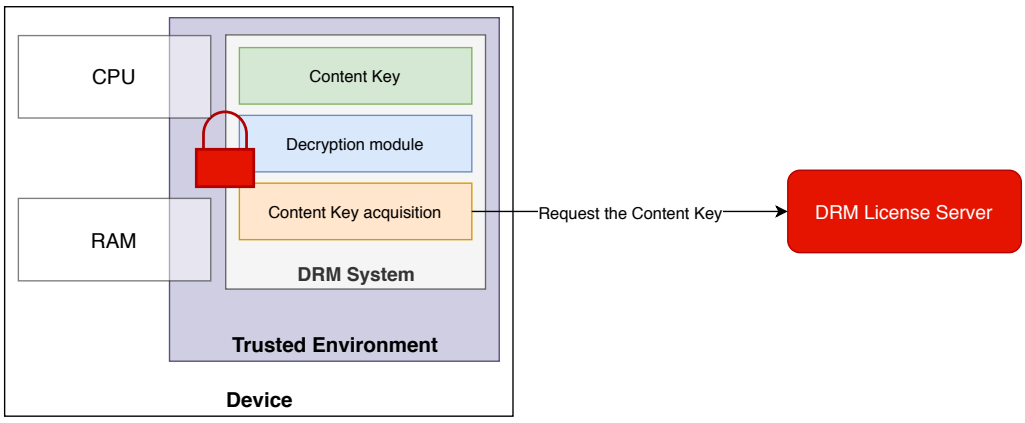

Figure 1.4: Simplified vision of a DRM system. The device has a trusted environment in the CPU and in the RAM which stores the Content Key fetched from a remote server, and then performs the decryption in a secure way.

Figure 1.4 shows a simplified view of a DRM system in a device. The DRM will create a secure and trusted environment within the device, the RAM and CPU usage of the DRM is encrypted and protected, so no other processes running in the device can read the data stored there. Then, the DRM will be responsible to call the external DRM license server requesting the content key of the stream, once returned, the content key must be used to configure the decryption module. The decryption module is in charge of decrypting the movie content without exposing any data of the decryption key. The DRM implementation in the device must match with the DRM license server, both technologies are proprietary and they depend on third parties which develop right protection mechanisms. The principal companies that offer Digital Rights Management solutions are Google, Microsoft and Apple, deeper information about DRM systems will be explained in the following chapters.

1.1.5. Playback session

At this point the streaming packages are properly set up and the company has enough capacity to stream content. Now it is time for the customers to enjoy the movies at their devices. Remember that each package has multiple versions of the same video which are named renditions, and each rendition is encrypted. The player, at this point, will fetch the streaming manifest and from that file it will select the renditions that are prone to be used. Apart from this, it requires a decryption key to be able to decrypt the content and start showing properly the image. The information on how to obtain the key it is also present in the manifest.

The player will request to a license server (key server) the content key to decrypt the files through a DRM system. The DRM system explained above will make sure that the key is be delivered in a safe manner so it stays secure at all points of the transaction. Once the device is able to decrypt the content, it will start fetching the data from the selected renditions through the CDN. Remember that the renditions have the same content with different qualities, the player can switch between renditions at any time, most probably because of network conditions.

1.2. Technical introduction

There are some technical topics that should be explained to properly understand the following chapters. These are related to the treatment of video data and with some encryption algorithms and patterns.

1.2.1. Codec

A codec in the multimedia world is an algorithm or method to encode and decode audio or video information (data). Usually they offer a standardized protocol to compress the stream, making it more efficient for storage or streaming. Codecs evolve according to capacity requirements, when the data increase on terms of quality, and quality derives to size, a new codec is designed, so the data can still be stored, treated and decoded on an efficient way.

This thesis is not focused on the codecs itself, but as an introduction for future references the most common codec for video streaming is H.264 or AVC (*Advanced Video Coding*). It is designed for video and is supported on the bast majority of devices. All major browsers can decode and handle H.264 streams.

Current codecs optimize video streams by storing the picture motion between frames instead of storing the full frame, it is known as motion compensation. This obviously reduces the amount of data that needs to be saved. In general, there are three kinds of frames, the *Intra Frame (I-Frame)* that is a complete representation, the *Predictive Frame (P-Frame)* that contains the motion-compensated difference between previous I-Frames. And finally the *Bipredictive Frame (B-Frame)* that stores the motion-compensation between past and future I and P-Frames. A *Group of Pictures* (GOP) starts with an I-Frame, and then contains a series of P and B-Frames. Figure 1.5 shows a very simple GOP with their frame references.

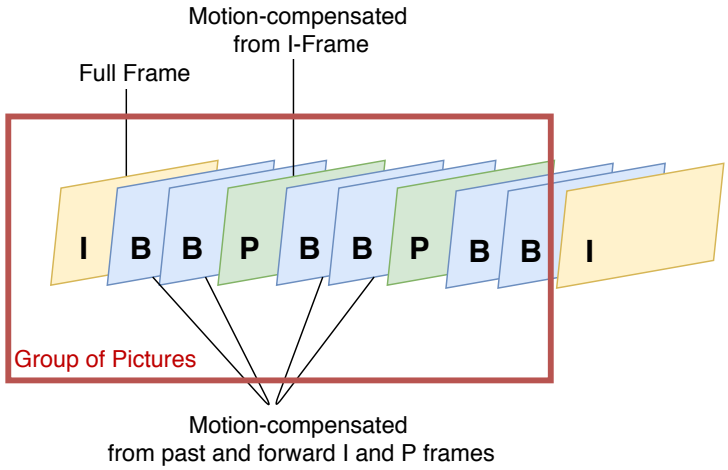

Figure 1.5: Frame sequence showing how a GOP works for video streams.

On streaming, the most important frame is the Intra-Frame (I-Frame or Key-Frame), a GOP always starts with one of those frames, which is required to successfully decode the whole group. Imagine a user that is watching a movie and at a certain point it decides to skip a scene. The decoder will seek an I-Frame near the position where the user moved, and from that I-Frame the player will start decoding the following P and B-Frames.

Last but not least, there is a concept of bitrate related to codecs. Bitrate is directly related to how much information is contained in a span of time. Encoding an image preserving a good quality detail will produce a high-bitrate video. Stream size will be much bigger but the quality will be good. If the codec encodes the image skipping details, and loosing some picture quality, the output stream will have lower-bitrate and the resulted stream will be lighter.

More bitrate requires more internet throughput, this has a direct impact on the streaming session. If internet's speed is lower than the bitrate, buffering events will happen on the client side. This is why current streaming protocols use bitrate ladders on the renditions, with them the customers can enjoy the best picture quality supported by their network speed.

1.2.2. Container

As it has been explained, a codec is a way to process and optimize data. But this data does not have any kind of format, it needs to be stored properly and this is what a container does.

A container gives format to the output of an encoding step, a codec generates raw data that needs to be stored and formatted so players can be able to decode it back to pictures and represent the image. Containers define the way this data is stored by a group of bytes that hold the information of the stream and the raw data. Containers usually store information of the codec and the encoded data, such as the bitrate, resolution, content type, etc. It basically gives format and persistence to raw data. They usually let specify the codec of the contained data, so players can decode it, but sometimes it might happen that a container does not support a codec because there is no way to specify it.

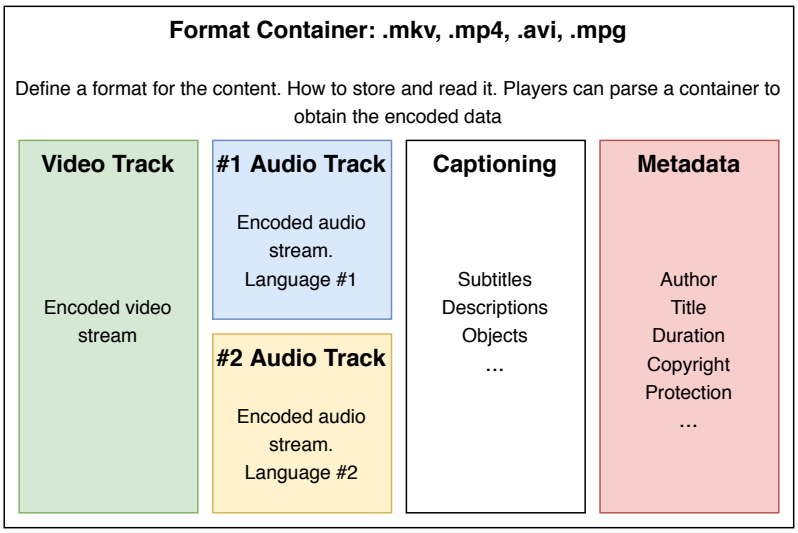

Figure 1.6: Container representation where different tracks of data are defined.

There are multiple containers on the market, this thesis will mainly focus on just one container the one that is called *MPEG4 Part 14* (MP4), but *MPEG-2 Transport Stream* (MPEG-2 TS) will also be relevant in this document. Both of them are suitable to store and stream through network audio and video content.

- **MPEG-2 Transport Stream (MPEG-2 TS)** was designed back into 1995 and it was built taking in mind the diffusion of the content through unreliable networks such as Digital Video Broadcasting (TV and Satellite). This container holds small payloads so it can recover from errors on streaming because information is segmented in small chunks. One of the streaming technologies that will be explained in this document started using this container, but recently it was deprecated in favor of MP4 because newer codecs where not supported. Refer to VideoLAN [2] website to know more about the supported codecs.
- **MPEG-4 Part 14 (MP4)** was standardized in 2003. It is a modern and flexible container which supports any kind of content such as video, audio or subtitles. It was

designed to support metadata within the container. Its first definition was done under the name of *ISOBMFF*. The container is structured in fields that hold information, these fields are known as Boxes (previously where named atoms), they describe and signal information of the container, tracks, content, etc. Some boxes hold timing information, others the stream size or even pointers to the data, so players can easily seek without having to read all the data content. MP4 compared to MPEG-2 TS can hold any codecs inside it, it just requires to signal the codec brand in the appropriate box. Finally, this container support streaming through HTTP but it has some issues when seeking, to solve this, there are containers based on MP4 such as fMP4.

– **Fragmented MP4 (fMP4)** is an evolution of MP4 container. The basic idea is to chunk data in small fragments which can be decoded independently. For example, instead of holding 2 hours of video data in a single MDAT (media data) box the video will be split in chunks of 2 seconds and each chunk will be stored in a different MDAT. This would result in a MP4 with 3600 MDAT boxes (also known as segments or fragments).

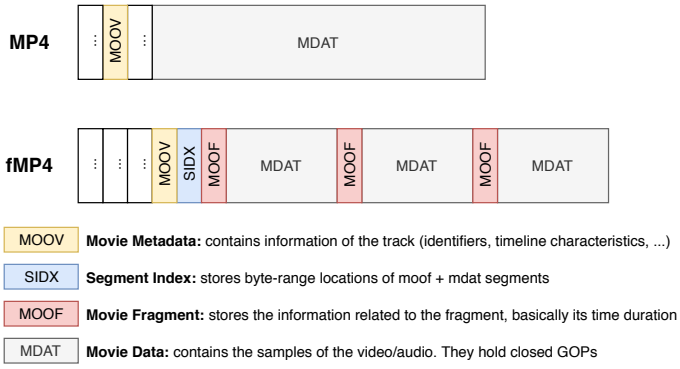

Figure 1.7: Difference between a regular MP4 and a Fragmented MP4. Notice the segmentation of the MOOF and MDAT boxes.

1.2.3. Encryption algorithms

One of the hot topics in this thesis is the multimedia protection. Protection is typically achieved by encrypting the content of a container. The standard algorithm used for media encryption is *Advanced Encryption Standard* (AES).

AES is a symmetric-key block cipher which accepts three key lengths: 128, 192 and 256 bits, the same key is used both for encryption and decryption. The algorithm uses a 128 bit block size, which means that the cipher can only process 128 bits at once, data will be split in blocks of 128 bits (one per operation).

To increase security, cipher blocks have different working modes so the key or the data changes at each step, same input data will be different on the output (data is randomized). This changes helps on avoiding repetition of data so the same input produces different outputs. The main problem is that changing the working mode also changes the ciphered output., same mode must be used for encryption and decryption.

There are several modes of operation but the most used ones in media encryption are

AES-CTR (Counter) and AES-CBC (Cipher Block Chain). Below there is a little explanation on how encryption works for both modes, just to explain that the output of will differ between modes.

- **Counter (CTR)** requires the use of a *Nonce* and a *Counter*, this data will be piped into the cipher box, so they must have a length of 128 bits. Then, the key is applied to the cipher-box and the output is XORed with the plain data also having a length of 128 bits. This process is repeated per each chunk of data and the Counter is increased by one, the Nonce remains the same. Figure 1.8 shows how the Nonce and Counter are used within the block cipher and how the output is XORed with the plain data.

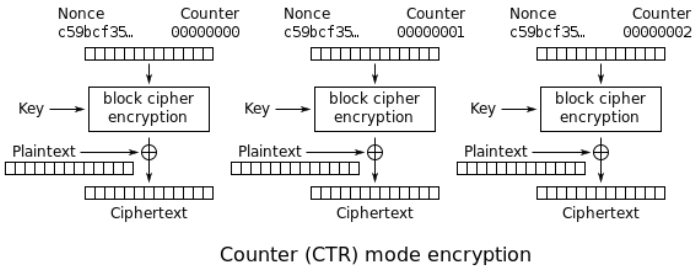

Figure 1.8: Encryption diagram using AES-CTR mode.

- **Cipher Block Chaining (CBC)** makes use of an Initialization Vector (IV) in the first encryption, this IV is XORed with the plain data and the result of this operation is piped into the cipher box which is encrypted using the key. The result of this operation will be the IV for the next encryption step. Figure 1.9 shows that each output is chained to the next operation, this randomizes the output even if the input is always the same.

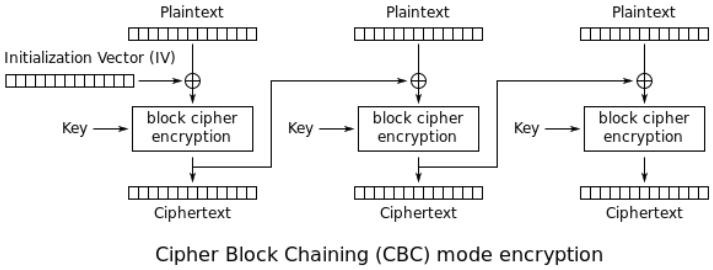

Figure 1.9: Encryption diagram using AES-CBC mode.

Refer to *Recommendation for Block Cipher Modes of Operation* [3] provided by *National Institute of Standards and Technology* (NIST) for more information about encryption or decryption using different working modes for block ciphers.

1.3. Motivation behind the Common Media Application Format

On the next chapters multiple streaming technologies will be show cased. But in a general, the current market technology is fragmented in several ways. There are multiple streaming protocols, and each one can use different codecs and containers. There is not a common standard that unifies streaming protocols, codecs and containers. On terms of protection, there are multiple solutions in the market that use either AES-CTR or AES-CBC, most of them proprietary. It must be said that, although the licensing (key exchange) is kept private and proprietary, all the solutions try to share the same underlying encryption schemas. In most cases, it depends on the device vendors to choose which decryption algorithms supports the device.

1.3.1. Fragmentation in RakutenTV

RakutenTV, as other VoD and OTT companies, suffer from fragmentation on which standards are supported by the different devices and partners. Its service is based on providing films and content to end users trying to support as many devices as possible. The company currently uses two streaming packages which are enough to cover all the major devices. The first one is based on a deprecated technology from Google, it is maintained due to legacy reasons. Most of the devices prior to 2013 do not support current streaming technologies, because of this the service is provided through this legacy technology. For the newer devices, there is a single package supporting two of the three current streaming technologies. On the state-of-the-art chapter the streaming technologies will be presented.

Figure 1.10 shows the two packages that are being used nowadays in the company. None of them can be used within Apple devices, Figure 1.11 shows the package schema that it would be suitable for iOS and MacOS.

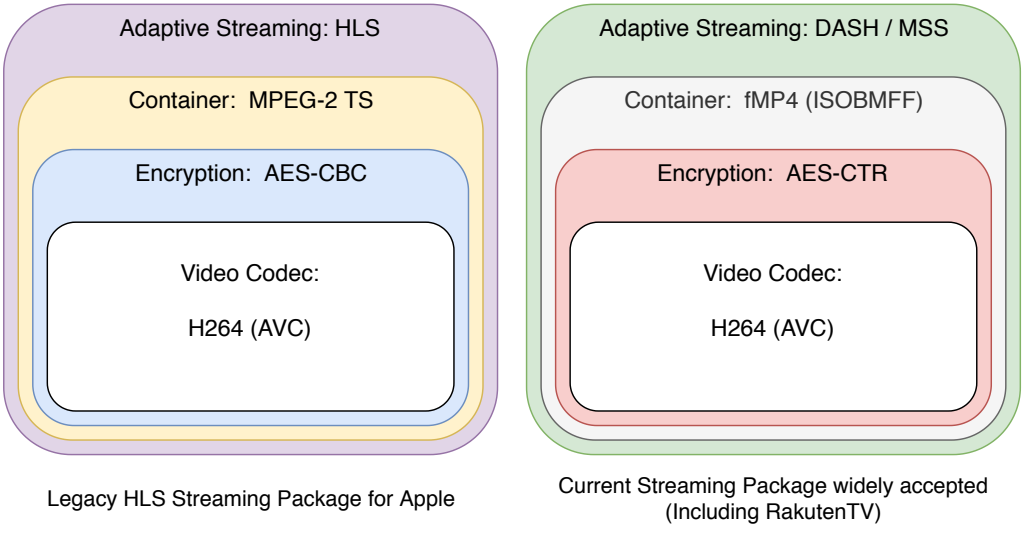

Figure 1.10: RakutenTV current packages. Supporting Widevine Classic (Deprecated) and DASH/MSS with Common Encryption.

Apple devices require another kind of streaming package, which instead of use AES-CTR it employs AES-CBC. The container is also different, it employs an MPEG-2 TS while the codec remains the same. Figure 1.11 this package:

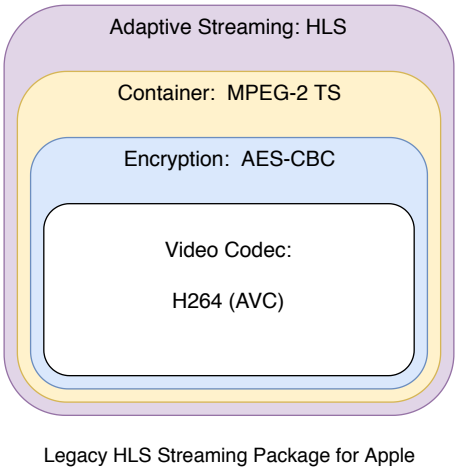

Figure 1.11: Apple Legacy Package to support iOS and MacOS devices.

Due to these technical discrepancies support Apple devices will imply the duplication of the whole library. Smartphones are not a priority for the company, so no investment is expected to generate the packages that can only be used in this platform.

1.3.2. Common Media Application Format. Solution to the fragmentation

To solve fragmentation issues, the industry of streaming together with device vendors are trying to unify all the technologies that build up a streaming service. Some of them are big players of the streaming such as Adobe, Akamai, Microsoft, Apple and Netflix, while others are device manufacturers like Lg, Samsung and Sony.

The new proposal is known as *Common Media Application Format* (CMAF) and defines the container and the encryption protocols. It has been designed to support both DASH and HLS streaming manifests. CMAF defines a new container format inherited from fMP4, this is why most of the research done in this thesis (and even the software tools in the market) assume that a fMP4 is a valid container for CMAF testing. Finally, the proposed encryption algorithm is AES-CBC. As it has been said, Apple has been involved in the definition of this format since they added support for fMP4 in their HLS streaming protocol.

Figure 1.12 shows the new composition for the CMAF package. It is a mix of streaming technologies between the used by Apple and the other ones used in the streaming sector.

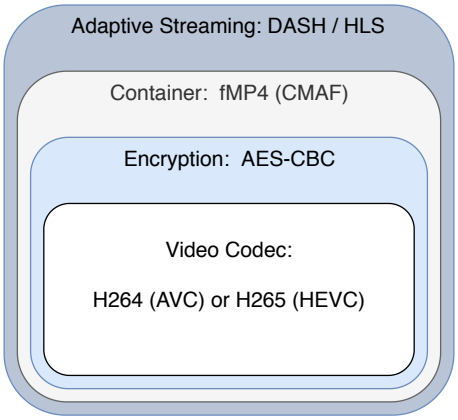

Figure 1.12: CMAF proposes a standardization on the container and encryption level. All the streaming parties agreed on a common format.

Through the development of this thesis what is going to be tested is the feasibility of CMAF. If it is possible to generate CMAF packages on a simple way and which is the current device acceptance. According to different information obtained from white and black papers or even in some conferences it seems that the vendors are slowly moving towards the new proposal.

1.4. Objectives

The proposed objectives are listed below:

- **CMAF packages:** to be accomplished during the first part of this thesis.
  - **Streaming packages:** generate HLS and DASH streaming packages using the fMP4 container.
  - **Use AES-CBC:** the streaming medias must use the new AES mode proposed by CMAF.
  - **Use content protection systems:** support the industry accepted content protection systems that work with AES-CBC.
  - **Validation:** summarize multiple devices that may or should support CMAF and validate if the built packages can be played properly on the devices. Those should support at least one of the included content protection systems.
- **Operational benefits:** on the second part of the thesis, benefits and cost savings of the new package will be studied.
  - **Package efficiency:** analyze the efficiency of adaptive streaming packages when multilanguage is used compared to not using such efficient packages.
  - **Viability of CMAF in terms of costs:** study the viability of CMAF within the company, which are the expected costs and how the company could support CMAF without big budget increments.

1.5. Document structure

In the following chapters deep explanations will be done starting with the state of the art, where current streaming technologies are showcased. It is followed by an analysis of current software that help on the realization of this thesis. Then, the next chapter introduces the validation of the hypothesis. Last but not least, the two remaining chapters will summarize the benefits of the solution and introduce some overall conclusions and feasible future work.

CHAPTER 2. STATE OF THE ART

After doing the introduction to the contents of the thesis and the different important technical topics, it is time to start with the state of the art in streaming. In this chapter current streaming protocols, encryption patterns and protection solutions will be introduced.

2.1. Streaming protocols

Streaming protocols and technologies have evolved over time. Although there are lots of ways to stream content, this thesis will focus on streaming over HTTP. There are two ways to stream content, the first one is *Progressive Download* which can be considered as a regular file download. The evolution of progressive download is *Adaptive Streaming* which focuses on streaming responsiveness to network conditions, player screen sizes and in general improve playback quality.

2.1.1. Progressive download

Progressive download can be seen as a regular file download where the player can start playing the video as soon as it has the enough bytes to start showing frames. This kind of streaming takes advantage of the HTTP/1.1 protocol that offers features such as requesting partial content and byte ranges.

There is some controversy when saying that this method is streaming, many parties consider it as a pseudo-streaming technique, because it enables to playback while downloading. But things such as skip or trick playing have inherent problems related to the container and the way data is sent. When doing trick playing, the player needs to know the position of the bytes that correspond to the selected time. This information is held by the container which normally have an atom that relates both time and byte position. The secondary problem is the codec itself, if the GOP is not constant the player needs to seek an appropriate I-frame so it can start playing.

Usually this kind of streaming use single files and do not accept adaptive quality. If the player or the network cannot cope with the bitrate requirements the play will start buffering. It does not adapt to the different network or device conditions, resulting in bad playback sessions.

RakutenTV use this kind of streaming for legacy reasons, old devices that do not support adaptive streaming can play files using this method. There are solutions for content protection too, one of them is Widevine Classic. This format defines a single container that can hold multiple streams and qualities, so the player can switch between them, it is not considered adaptive streaming at all because the protocol is strict on what it offers, for example it does not permit multi-resolution streaming, it won't adapt to multiple screen sizes.

Progressive download has issues when adapting to network or device conditions. Devices suffer from buffering because sometimes the network throughput is lower than the video bitrate, and the Widevine Classic package cannot lower the resolution so the playback

session will start buffering.

2.1.2. Adaptive streaming

Progressive download evolved to *Adaptive Streaming* to improve the quality and performance of streaming. Adaptive streaming protocols have been designed to be aware of network and player conditions, for example they support multiple resolutions and bitrate qualities. A single adaptive streaming package can contain versions of the same video in multiple resolutions being able to adapt between devices. It is not the same a SmartTV with a 60-inch screen than a 4-inch Smartphone, but both devices can be covered using an adaptive streaming, the player will select the appropriate resolution. These packages do not only have single resolutions but also different bitrates per resolution, depending on the network capacity the player can select the most appropriate video between all the available options.

Being able to adapt to network conditions will improve the quality of experience perceived by the customer. Buffering will decrease because the player will be able to select the lower qualities, but if the internet connection has good throughput the user will be able to play the higher definitions.

An adaptive streaming package can be seen as a video that has been encoded in multiple versions of itself, then a manifest holds all the information related to the encoded files. The players will use the manifest to know the characteristics of each video, so when performing the playback it can switch between all of them.

Figure 2.1 shows an example of three different renditions in a streaming package. They are using a fragmented container, for example the fMP4 explained in the introduction. When a container is fragmented all the resulting fragments will hold data with same duration, so the first fragment of the first rendition will contain exactly the same samples of the first fragment on the second rendition. To simplify things, lets focus only in the MDAT box of the container, it contains the encoded video or audio, in case of video it is a closed group of pictures. As it has been explained on the project overview, each GOP starts with an I-Frame that is a full image frame. With a fragmented container a player can easily switch between renditions only by knowing which is the following fragment to be shown. Each fragment contains the enough video data to decode and show, fragments are independent units of data, they can be processed alone. On the industry it has been standardized segments of 2 seconds as a general rule, although different time lengths can be used. This means that each MDAT will contain the required bytes to show 2 seconds of video.

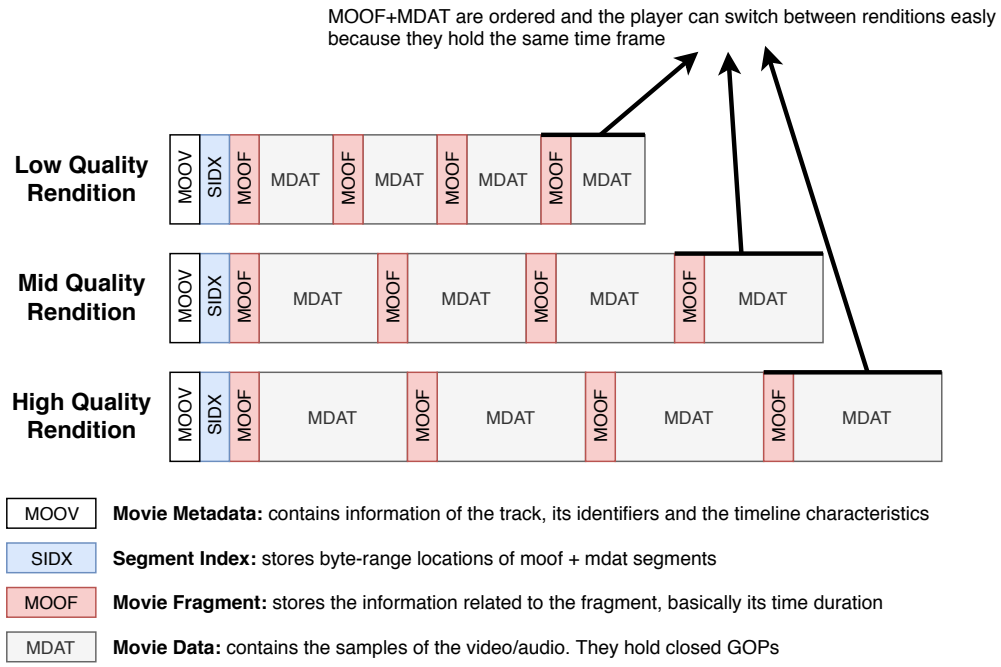

Figure 2.1: Three different renditions using an fMP4 container. Each rendition contains the same number of segments holding a similar duration between them. Fragmented MP4 makes easy the quality switching because of each fragment is independent of the others.

There are multiple *Adaptive Streaming* technologies, considering the case-study of this thesis lets dive into the most relevant ones. On the following sections Smooth Streaming, DASH and HLS protocols will be briefly introduced.

2.1.2.1. Microsoft Smooth Streaming (MSS)

Microsoft designed this streaming protocol as an extension for their Internet Information Services (IIS) to provide streaming support on the media services. It was built on top of HTTP and defined its own container format called *Protected Interoperable File Format* (PIFF). Fortunately PIFF was based on *ISO Base Media File Format* (ISOBMFF) which is the standard behind the MP4, this also makes fMP4 is a suitable container for MSS streaming.

Smooth Streaming has some limitations inherent to how it was designed, it restricts the codecs to be used inside the container so newer encoding algorithms are not supported, this makes device vendors to start supporting other streaming protocols. For example, when streaming Ultra High Definition videos there are several codecs that compress a lot the video and are becoming the industry standard for such content, the main problem is that these newer codecs cannot work with MSS. Because of this, Microsoft has decided to deprecate this protocol in favor of DASH.

Smooth Streaming clients are not required to support ISOBMFF. Players are provided with a manifest file containing all the video and audio qualities (resolutions and bitrates) and their respective time segments. The player will query a transmuxer that will convert a time segment to a byte range and return that byte range from the rendition specified by the client.

Figure 2.2 shows an example of this kind of streaming, the transmuxer is a required element that converts time segments to byte ranges. With the transmuxer devices do not require support for the underlying container, they only need to support the codec which is the data being returned by the transmuxer.

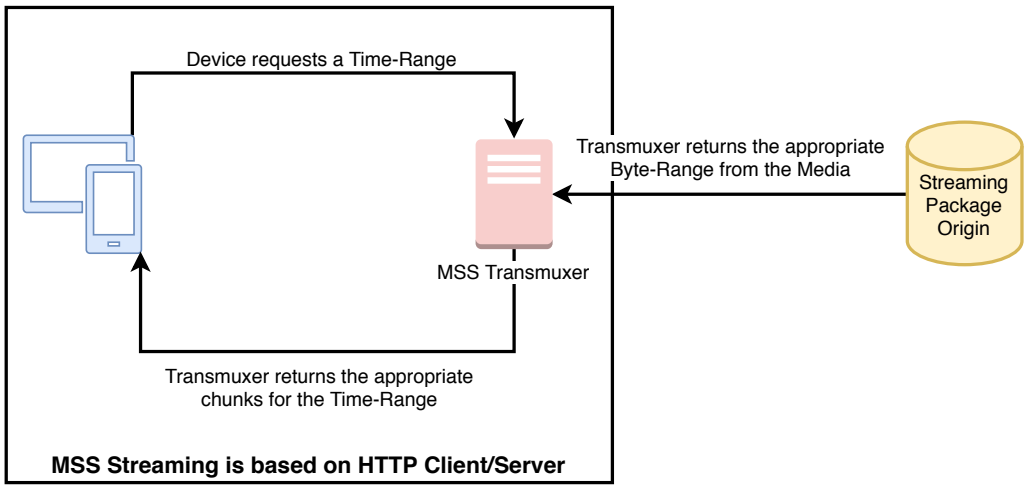

Figure 2.2: Microsoft Smooth Streaming is based on transmuxers which convert the requested time range to the corresponding data chunk.

Appendix A. *Adaptive Streaming Manifests* in Section A.1.*Microsoft Smooth Streaming* client and server manifests examples are shown and explained how they are used.

Finally, when it comes to content protection, the standard defined only support for AES-CTR. PlayReady DRM was the only approved DRM system on this streaming technology. Head to section *2.3. Content key acquisition* for further information.

2.1.2.2. Dynamic Adaptive Streaming over HTTP (DASH)

Most of the knowledge acquired in the development and operations of Smooth Streaming was used when defining the *Dynamic Adaptive Streaming over HTTP* protocol. DASH (also known as MPEG-DASH) was designed from scratch to support adaptive streaming between an HTTP server and a client without needing a transmuxer. Clients are powerful enough to read a container and request the data chunks. It has been standardized under the *International Organization for Standardization* (ISO) in the specification *MPEG-DASH ISO/IEC 23009-1:2014*. It makes use of fMP4 container to hold the video data.

DASH manifest contains all the available renditions with its technical information, on each rendition entry the file location and the byte-ranges for the initialization segments in the container are specified. Being said this, the client will download the headers of each fMP4 container where the initialization segment is located, with the initialization segment the player can calculate the position of each fragment for a specific time interval. The player can switch between renditions because all of them contain the same amount of fragments. In this case, because of the SIDX box that translates time to byte position there is no required transmuxer between the player and the origin files, they are reached using byte-range requests.

Figure 2.3 represents the difference with the previous picture 2.2, in this case there is no transmuxer between the player and the media.

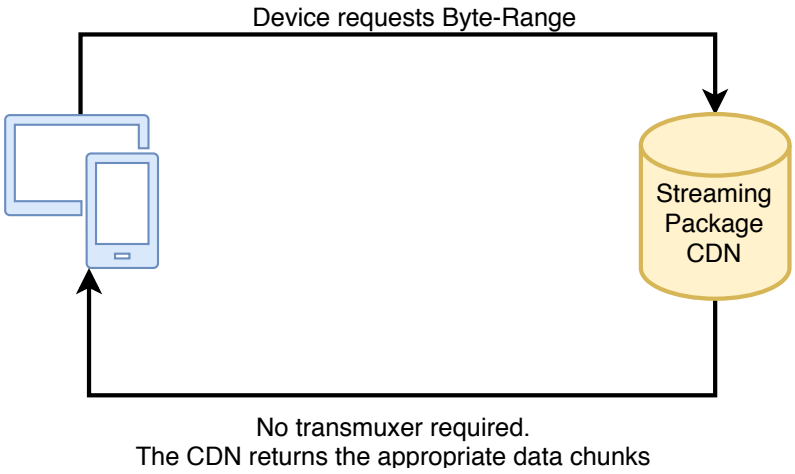

Figure 2.3: DASH does not require a transmuxer. Devices are smart enough to request the appropriate byte-ranges.

Appendix A. *Adaptive Streaming Manifests* in Section A.2.*Dynamic Adaptive Streaming over HTTP* shows an example of a DASH manifest for VoD.

The same organization that standardized this streaming protocol designed an encryption system to work on top of it. *Common Encryption* was designed for ISOBMFF containers to provide content protection. The same standard provided information on how to signal the encryption data on DASH manifests, so both protocols are really suitable for streaming VoD content. Head to Section 2.2. Common Encryption to know more details about media protection.

The main drawback of this technology is the support on Apple devices, protected DASH streaming is not supported in iOS and MacOS.

2.1.2.3. HTTP Live Streaming (HLS)

Finally, HTTP Live Streaming was designed by Apple to be integrated within its software and hardware. It is similar to MPEG-DASH based on plain HTTP requests. In this case Apple submitted an RFC draft, the latest version (v7) is RFC8216 [7].

First releases of this protocol only supported MPEG-2 TS, as it has been explained it cannot store data from newer codecs. Fortunately, in the version 7 of HLS support for fMP4 was included. HLS manifests are similar to DASH, players can handle directly the container and obtain the appropriate byte-ranges from an HTTP server, no intermediate transmuxer is required, it works on the same way as Figure 2.3.

HLS manifests are split into multiple files, the first one holds the information of all the renditions, each entry points to a secondary manifest related to a single rendition file. In that manifest all the byte-ranges are listed so the player can fetch from the origin the appropriate bytes. To know more about the manifests please check Section A.3. HLS in the appendix.

Last but not least, HLS supports content protection using multiple DRM systems, although the most relevant ones are Widevine and FairPlay. FairPlay is the one that will work in Apple devices and it makes use of Sample-AES for the underlying encryption. Sample-AES is basically an AES-CBC with pattern encryption. Next section will cover the details of encryption and their modes.

2.2. Common encryption

Content providers and cinema studios are very strict on the treatment of movies, they enforce strict rules on the content protection. All the streaming chain must be secure enough so the content does not get leaked and widely available for free.

Back in 2012 there were efforts from multiple parties to standardize under the *International Organization for Standardization* (ISO) a specification for media encryption using MP4 containers. It was standardized under the name *MPEG - Part 7: Common encryption in ISO base media file format files* (ISO/IEC 23001-7). The specification was revised in three different editions being the last one published on 2016. Each revision added new encryption mechanisms and signaling metadata for the container.

Common Encryption specifies a standard encryption and key mapping method that can be used to secure multimedia files. In this section the focus will be the encryption, the decryption side will be covered in the next section (*Content key acquisition)*.

The standard makes use of *Advanced Encryption Standard* (AES) and support two encryption modes, AES-CTR and AES-CBC, both of them can be used to protect the content. Within these modes there are multiple scheme types, that define how the video or audio samples must be encrypted.

2.2.1. Scheme types

AES modes are CTR which is based in a counter and CBC which chain blocks by using the encrypted result. Within these two modes the ISO standard also introduced the encryption scheme that defines how the content is going to be encrypted. The content can be fully or partially encrypted depending on how video samples are treated:

- **Full Encryption:** when applying this scheme all the video (or audio) samples inside the MDAT container will be encrypted. No content will be left on the clear.
- **Pattern Encryption:** this second scheme type appeared to improve the performance of decryption on low end devices. Instead of performing a full encryption of the content, it partially encrypts the data stored in the container. The samples (or frames) will be encrypted using a pattern, some samples will be encrypted and other ones will be left in clear. This reduces drastically the amount of data that a processor needs to decrypt, the rationale behind this scheme is that it improves the decryption throughput as fewer data needs to be decrypted. Over time resolutions are being increased which means that each frame will hold much more data, improving the efficiency on the decryption will let devices with less resources still work with higher resolution content.

As it has been said, both scheme types can be applied to the existing AES modes. The combination between the encryption mode and the encryption scheme is represented in Figure 2.4, each combination receives a name that is used to identify the scheme and mode used to encrypt a stream. This name is signaled in the streaming manifests and inside the container.

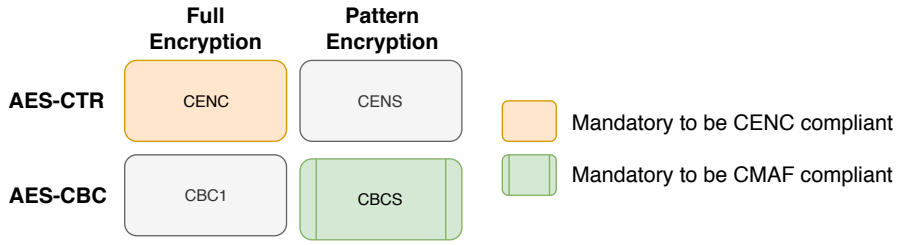

Figure 2.4: Scheme types included in the Common Encryption definition by MPEG - Part 7: Common encryption in ISO base media file format files.

First release of the Common Encryption specification only defined the *cenc* type and it was mandatory to be supported by any system implementing such definition, on subsequent standard updates the other combinations where added.

Common Application Media Format enforces the usage and support of AES-CBC with pattern encryption. Every device that properly supports CMAF will be required to support the encryption scheme *cbcs*.

2.2.2. Signaling the container

Medias and manifests must be signaled with the encryption information or players will not be able to decrypt and play the content. Common encryption enforces the signaling of multiple parameters, the most important ones will be the key identifier (which links a track with an encryption key), the scheme used when encrypting and finally the DRM systems supported within the media.

The following bullet points summarize the most important parts to be signaled within the media or the streaming manifest:

- **Key identifier (KID):** encryption keys must be associated to the corresponding video or audio track, so the player can request the appropriate decryption key. KID is the key identifier and it is written both in the container and the manifest, the player will use this KID with a content key acquisition system to fetch the key and decrypt the stream.
- **Encryption scheme:** the scheme must be signaled so the player can configure the decryption mode on the module in charge of decrypting fragments. On common encryption AES uses 128-bit keys, but the underlying algorithm and scheme type works different depending on the mode to be used. The appropriate scheme name from the four defined scheme types will be used to signal the encryption type both in the container and in the manfiest. The valid identifiers for the schemes are *cenc, cens, cbc1 and cbcs*.

- **Protection System Specific Header (PSSH):** the medias and manifests must be signaled with the specific DRM system protection as Common Encryption defines. Each DRM system is recognized by a unique identifier which can be specified either on the appropriate fMP4 box or in the manifest itself. Within the system identifier some other data is included which is used by the player to engage the DRM protection.

2.3. Content key acquisition

After explaining the different streaming protocols and content protection methods it is time to introduce how the system obtains the key to decrypt the content. This step is known as content key acquisition, players can obtain the decryption key using several methods, one of them is defined by the common encryption standard and it is mandatory to be included in any browser following the CENC spec. Apart from the key acquisition method defined on the standard there are other proprietary methods known as *Digital Rights Management* (DRM).

As it has been said, the manifest and the container are signaled with the KID, encryption scheme and PSSH. Players will use this data to query an external server and obtain the decryption key for the specified KID. With that key and the encryption scheme.

On the following subsections ClearKey and different DRM systems are introduced, although the first one is not suitable for production environments it is good as a starting point in media encryption.

2.3.1. ClearKey

ClearKey is the reference implementation that any browser should support when implementing common encryption and the *Encrypted Media Extensions* (EME). This was done to ensure that open-source browsers had at least one open protocol to acquire content keys. Usually open-source community is not prone to include software that restricts users liberty, and that is what a DRM system does. Most open-source browsers will not have a DRM bundled in it unless they agree to do so.

When using encrypted content the websites will use the *Encrypted Media Extensions* (EME) included in the browser, they require some signaling to engage all the subsystems. In the case of ClearKey the system will respond to the usage of *"org.w3.clearkey"* identifier. When using that identifier the browser will request the content key for the supplied media KIDs (usually, in ClearKey there is a method to supply the content keys). Once the system is engaged with the appropriate content keys the decryption module will decrypt the buffered video samples and image will be shown.

This method is not secure enough for content providers because it does not ensure the privacy of the content key. A user with average programming skills will be able to intercept the calling methods on the website code and leak the decryption keys. Leaking the decryption keys means that content will be easily decrypted and available for piracy in matter of minutes.

2.3.2. Digital Rights Management (DRM)

Studios enforce strict measures to keep the content secure, this means that users are granted to watch the content but cannot hold the content in clear, to avoid this it is important to keep the content key private, even for the users who bought the video. Any device included in an OTT platform with copyrighted content must support a secure content key acquisition system that keeps the key hidden. *Digital Rights Management* (DRM) mechanisms are designed to enable the secure decryption of the content without exposing the key details. They are proprietary solutions embedded in the browsers or operating systems, and they work by creating trusted environment where the content key is stored and the decryption is done.

Modern DRM systems use the Trusted Execution Environment (TEE) embedded in the processors to create the secure environments that are used to store and process the copyrighted content. TEE provides an isolated environment that grants confidentiality and integrity to the processes that are using them. Only the process that created the isolated environment is granted to read the information stored there, no other processes within the CPU can obtain the content from there. But not all the devices have TEEs, in the case where a device does not have a secure environment the DRMs work by using software obfuscation. Most studios only allow lower resolutions to be reproduced when the security is provided by obfuscation.

Content key acquisition through DRM systems is known as *License Acquisition*. Within a license there is the content key and some play rights that are used to configure and manage the playback session. These rights are usually used to keep control of the playback session, for example, in a VoD service the DRM will control the rental time and the number of plays, if the content was rented for 24 hours once this time has finished the system will lock and no further playback will occur. There are other features in this play rights, most of them to control different security policies to avoid the copy of the content.

DRM is used through the *Content Decryption Module* (CDM) which is the standard interface that browsers (and other devices) offer to interact with the proprietary DRM systems. CDM's relay in their own private protocols which are not public, part of the security of the DRM is provided by the closed algorithms they use.

To briefly explain how license acquisition works, please refer to Figure 2.5 while reading this explanation:

1. **Generate a license request:** with the PSSH information in the container or in the manifest the player will engage with the DRM system. By using the *Encrypted Media Extensions* the player will use make the CDM build a license request. This request will contain the KIDs and some private information that the server will use to encrypt the key and securely return the license.

2. **Request validation:** when the license server receives a request it validates that the user/device who is requesting a license have rights over the content. If so, it will generate a license for the requested content.

3. **Generate license response:** as it has been explained a license is a bunch of information containing the content keys of the media and the play rights to be used in the player. Usually DRM systems encrypt the content key with the private information re-

ceived in the request. Most of the systems use certificates with public and private key, by using the public key secret content can be exchanged.

4. **CDM configuration:** when the CDM receives the license response it will engage all the policies defined in the license. From the response it will obtain the content key and configure it in the decryption module. The license acquisition and content key processing is done in the Trusted Execution Environment (if available) or in the software obfuscated environment. The information contained in the license will not be leaked to outer processes.

5. **Playback starts:** finally the CDM will start decrypting the buffered frames and image will show while more content is fetched from the CDN.

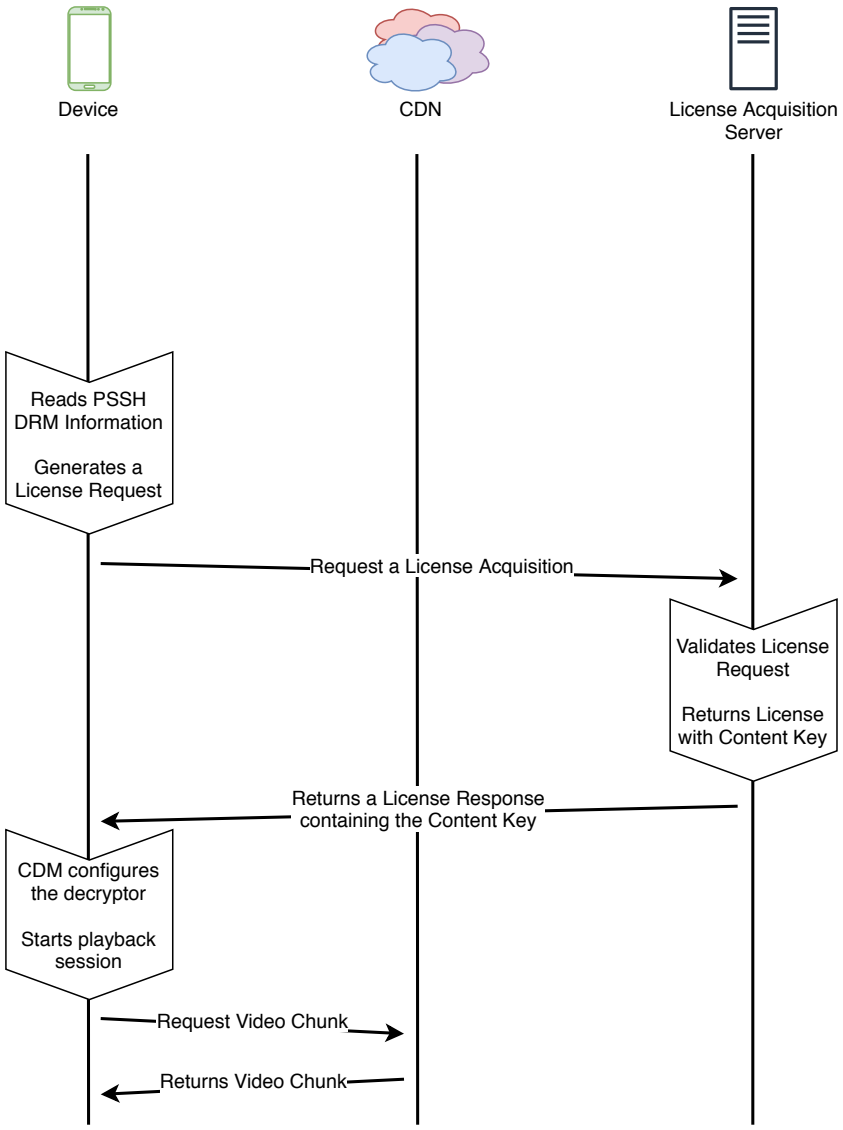

Figure 2.5: License Acquisition process on a Digital Rights Management system.

On the following sections the most relevant DRM systems for media and VoD are explained.

2.3.2.1. PlayReady

Microsoft designed PlayReady back in 2007 and it has been used since the appearance of Smooth Streaming. Fortunately it is compatible with Common Encryption as it started using AES-CTR and *cenc* scheme since the beginning. Its was design as a platform independent DRM, because of this, Microsoft released a *Device Porting Kit* that can be used to integrate the PlayReady DRM in any device.

It is the most widely used DRM solution on connected devices, almost all SmartTVs and gaming consoles support PlayReady, Windows has native support embedded in the operating system and there are solutions to integrate the DRM in Android and iOS applications.

PlayReady's CDM can run either at application layer or operating system layer, its security will depend on the usage of Trusted Execution Environment or not.

2.3.2.2. Widevine Modular

Widevine was a company bought by Google which started on the DRM licensing with Widevine Classic. At that moment they defined all the streaming stack, from the container to the plugin (similar to a CDM) running on the remote browsers or devices. They didn't used any kind of open technology and when Common Encryption was standardized they decided to deprecate it in favor of Widevine Modular.

Widevine Modular was built following the Common Encryption standard and was present since the beginning on Google products and devices. Since 2017 Samsung's SmartTVs added support for Widevine Modular.

On Android and Chromecast the CDM runs at the operating system level, ensuring a Trusted Execution Environment, but on browsers the secure environment is done by software obfuscation which is less secure. For example, Widevine CDM was recently broken in Chrome because it was using software obfuscation techniques and not reliable trusted environments.

2.3.2.3. FairPlay

Last but not least, FairPlay is the DRM solution provided by Apple and implemented in all the company devices. FairPlay is only supported on Apple devices, but because they have a big market quota for VoD services it is interesting to support it, customers are used to iPads to watch the platform content.

Although there are solutions to use Widevine or PlayReady in iOS they do not work as good as FairPlay. In the case of Apple's browser, Safari, it only supports FairPlay DRM. The CDM has been implemented at operating system level, which ensures the Trusted Execution Environment.

CHAPTER 3. TECHNOLOGY AND TOOLS ANALYSIS

In this chapter an analysis of the available tools will be done. When developing this project, there was an internal requirement to use only open-sourced code. Below different streaming packages, DRM systems and players will be listed and analyzed. They will be used on the implementation of this thesis.

3.1. Streaming Packager

On terms of streaming packagers there are two big players that open-sourced their code, *Bento4* and *Shaka Packager*. Both of them are suitable for this thesis because they support the required streaming protocols (DASH and HLS) and the appropriate encryption scheme (*cbcs*).

- **Bento4** is mainly developed and maintained by Gilles Boccon-Gibod, owner of Axiomatic Systems LLC. It is offered in a dual license system (open source and commercial). RakutenTV uses Bento4 because of its support for current streaming protocols, MSS and DASH are generated with this tool. In theory, it also supports HLS so it is a suitable solution for this project.
- **Shaka Packager** is developed by Google, which is creating a multimedia suite consisting of a packager and a player. Its streaming packager only support DASH and HLS, because of this it was not considered on RakutenTV. Now with the CMAF project it could make sense to test its viability, as it supports by default CMAF compliant packages.

There are other solutions in the market which provide the same features as Bento4 and Shaka Packager, but the problem is that they are not open source, or they are offered as *Software as a Service* (an application for encoding and packaging offered as a cloud service). Some of them are listed below:

- **Azure:** Microsoft's cloud provider offers unified solutions to package and deliver multimedia content. Support for CMAF has been added recently.
- **Unified Streaming:** this company offers solutions for media packetization including encoding, packaging and streaming. It has also added support for CMAF medias.
- **AWS Elemental:** Amazon Web Services have a suite of media products, they offer tools for packaging and streaming which they do already support CMAF.

Although they are well-known solutions in the market, trusted by several VoD companies these solutions are not suitable for RakutenTV. The philosophy of the engineering in the company is to use open source tools that can be extended and modified upon requirements. Sometimes teaks to the packagers must be done in order to support a wide range of devices.

3.1.1. Comparison of Packagers

Below a table summarizes the key features between Bento4 and Shaka Packager. They offer similar features and both are open source which make them suitable for this thesis. On the implementation chapter both packagers will be tested in order to verify that they can build CMAF compliant packages.

	Bento4	**Shaka Packager**
Streaming Packages	MSS, DASH, HLS	DASH, HLS
Encryption	Common Encryption (cenc, cens, cbc1, cbcs)	Common Encryption (cenc, cens, cbc1, cbcs)
DRM	PlayReady Widevine Modular FairPlay	PlayReady Widevine Modular FairPlay
Widevine Classic	No	Only decrypt
Platforms	Linux, Mac, Windows	Linux, Mac, Windows
Programing Language	C++, Java & Python	C++
Command Line Interface	Yes. Through Python	Yes. Compiled binary
License	GPL v2.0 non-GPL commercial license	BSD-3

Table 3.1: Comparison between principal open source packagers.

3.2. DRM License servers

Copyright content requires secure ways to protect and deliver the decryption key, in terms of DRM licensers the company manages their own licensing infrastructure, so license servers are built in-house instead of using third-party providers. PlayReady and Widevine Modular servers are developed and maintained by the Playback department. FairPlay has not been used in the company, but within the context of this thesis an evaluation of available options has been done.

DRM companies force signing *non-disclosure agreements* when companies use their software, the explanations of each system are a general overview without deeply entering in details.

3.2.1. PlayReady

Licensed companies are provided with a Server SDK that must be used to build the server, it contains all the necessary software to build an HTTP server that can handle PlayReady license requests, it automatically performs all the parsing, generation of license, encapsulation of content keys, etc. The developers in the company are only required to implement a method that will be used by the SDK. This method will configure the license params that should be used when issuing a license, in general they will be the associated content keys with the KIDs, and the play rights of the content (validation, license duration, output protection, etc).

Companies are forced to use the Server SDK programming language in their custom code because of it is integrated within the SDK calls.

Figure 3.1 shows a block diagram of a PlayReady license acquisition process, the green blocks are provided by the SDK while the blue one is implemented by the company.

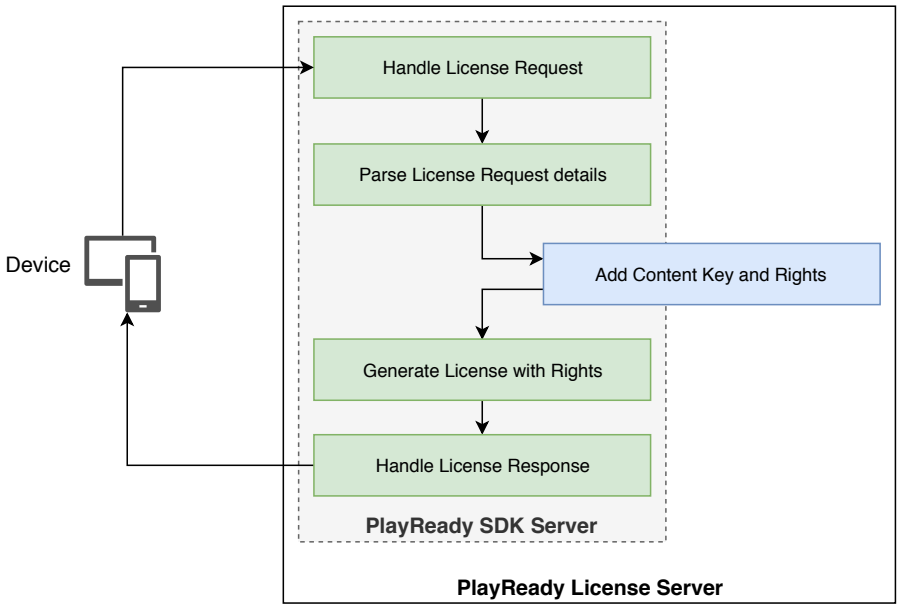

Figure 3.1: License Acquisition in FairPlay. The company uses the provided PlayReady Server SDK to build the service, it only needs to attach a small method that adds the content key and the play rights to a license.

Following the previous picture this are the steps carried out in a PlayReady implementation:

1. **Request handling and license parsing:** once the server receives a new license request the PlayReady SDK handles it, it parses the content of the license request and calls the implemented method with the parameters of the license request.

2. **License configuration:** this step is performed in the company code, basically it will use the parsed license to fetch the requested KID and return the appropriate content keys along with all the play rights a device should have.

3. **License generation and response:** finally with the license parameters the PlayReady SDK will generate a valid license which will be returned to the device.

3.2.2. Widevine Modular

Google took a different approach than Microsoft, instead of offering a Server SDK that builds a complete solution they offer a remote service which parses and generates the licenses. Companies are required to use that service through an intermediate proxy which wires the request validation and the license configuration with the remote service.

Companies can choose the programming language by their own, they only need to be able to perform HTTP requests to the remove Widevine service.

Figure 3.2 shows the basics of Widevine Modular license acquisition, the *Widevine Modular Proxy Server* is developed by the company (blue blocks) while the *Widevine Modular License Server* is offered by Google (green blocks).

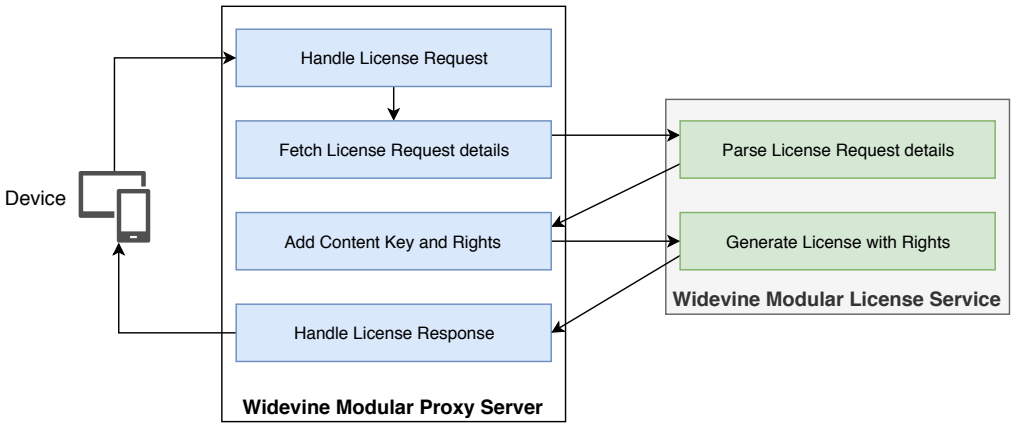

Figure 3.2: License Acquisition in Widevine Modular. The company builds a Proxy Server which uses Google's Widevine remote service.

Following the previous picture this are the steps carried out in a Widevine Modular implementation:

1. **Request handling:** the proxy receives a license request but it does not know how to read the license request details.
2. **Remote license parsing:** with the raw payload, the proxy requests the remote service to provide a parsed license request, with all the details in clear. This details will include the requested KIDs and general device information.
3. **License configuration:** once the proxy receives the parsed request it can generate the license parameters containing the content key and the different play rights.
4. **Remote license generation:** the license parameters are sent to the remote service which is in charge of generating a Widevine Modular license.
5. **License response:** the license generated by the remote server is directly returned to the device, the proxy only forwards the response.

3.2.3. FairPlay

Last but not least, Apple's server implementation is the most basic one, they deliver a package containing the source code of the cryptographic operations that are used to issue a license. The problem is that the source code is written in C which is not a common language for server side development. The C implementation is a reference example, Apple expects the companies to integrate the source in their systems or port the whole license code to another programming language.

There are two open-source implementations of FairPlay's code, one written in Java [13] and the second one written in Golang [14]. After analyzing both of them, the Golang one is much easier to use and modify.

Figure 3.3 shows how the licenser works, as the other examples, the blue boxes are developed by the company while green ones are provided by the FairPlay package.

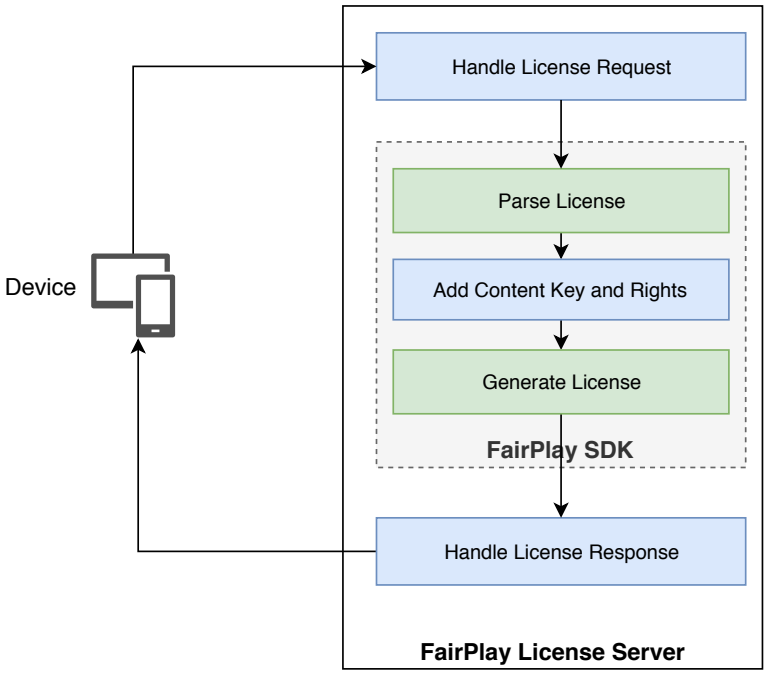

Figure 3.3: License Acquisition in FairPlay. The company builds the whole License Server and it calls the embedded FairPlay SDK with the appropriate license parameters.

Following the previous picture this are the steps carried out in a FairPlay implementation:

1. **Request handling:** the server receives a license request, using its payload the FairPlay method that generates licenses is called.
2. **License configuration:** Apple's algorithms will take care of parsing the license request and call a company-implemented method that returns the content key and the license parameters.
3. **License generation:** once the FairPlay SDK has the license parameters it can issue a valid license which is returned to the device.

3.3. Players

On terms of players, the idea was to cover all the major devices that RakutenTV supports. In this thesis the players and platforms that are expected to support the new CMAF package will be evaluated. The easiest devices to make them support the new standard are Web Browsers and Smartphones, whereas SmartTVs, Consoles and other ones, will be harder to make them work. Usually, manufacturers do not add new features in their firmware updates.

3.3.1. Web Browsers

Within the Web Browsers section, different browsers and players will be covered. The most relevant browsers, taking into account their support for DRM are Google Chrome with Widevine Modular, Microsoft Edge for PlayReady and finally Safari which employs FairPlay. In terms of players the big players are *DASH-IF* which is being developed by the DASH Industry Foundation and it is the reference player of DASH. The other option is Shaka Player, which is maintained by Google and it is widely used.

Browsers can use a DRM CDM with different strategies, some of them can include the CDM on the application layer, bundled with the browser, or others can use directly the CDM embedded in the operating system. Usually, browsers that come installed with the operating system will use the CDM embedded in the operating system, others browsers will be bundled with a CDM.

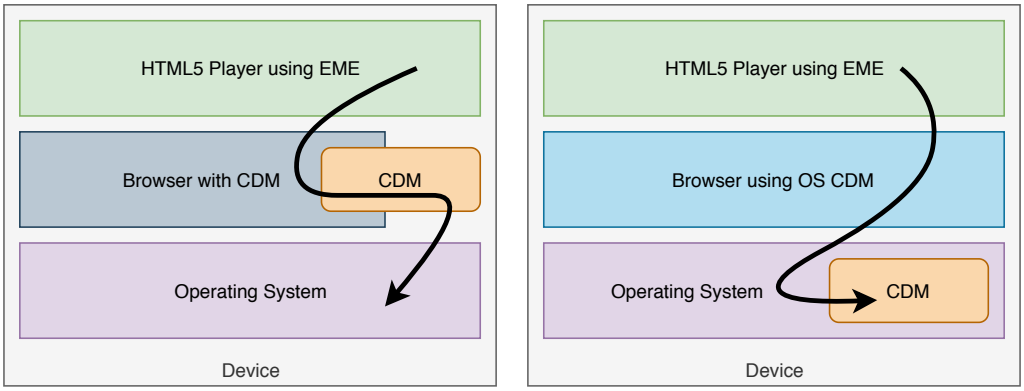

Figure 3.4: Browsers can use bundled CDMs in the same application or use the Operating System CDM. Chrome uses the first approach while Edge and Safari uses the second.

- **Chrome:** on this browser, the technology that will be tested is DASH with both players. Chrome is developed by Google and it includes the Widevine CDM. Since version 68, the included CDM supports *cbc1* and *cbcs* encryption. First tests where carried out on the *Chrome Canary* (alpha version with test features), on summer 2018. When this thesis was being written in 2019, CMAF package was already being supported in the stable branch of the browser. Chrome implements the Widevine CDM at browser level, so they can update the bundled CDM by releasing a new version of Chrome.

- **Microsoft Edge:** currently, in the company, DASH with PlayReady is being used in Edge browser. According to Microsoft, in the PlayReady Conference at New York 2018, it was confirmed that Edge will include AES-CBC support in future releases. The implementation of PlayReady in the client side depends on the Windows Core Team, instead of the Edge developing team. In this case, Edge does not bundle a standalone CDM but it uses the included in Microsoft Windows, the CDM operates at system level instead of browser level. Updating Edge will not modify the CDM version, it depends on Windows upgrades.

- **Safari:** this is the included browser in all the Apple devices. HLS is supported directly by the browser, it is able to parse and play *m3u8* manifests. Starting from

macOS 10.12 Apple introduced support for fMP4 container when streaming HLS medias. Encryption schema for Apple has been always AES-CBC with *cbcs*, in this case, Safari should support CMAF packages using FairPlay CDM. Apple follows the same strategy as Microsoft, the CDM is bundled in the operating system instead of being included in the application layer.

3.3.2. Smartphones

There are two big players in the Smartphone market: Android and iOS. Android supports Widevine Modular because its CDM is managed by the operating system, while iOS directly includes the FairPlay DRM.

- **Android:** Google introduced a new Widevine CDM version starting from Android 7.1 (Nougat). This new version enables the playback of AES-CBC encrypted content, supporting both *cbc1* and *cbcs* encryption schemes. CMAF requires the usage of *cbcs*. Any device using a version equal o higher than Android 7.1 will be able to play any CMAF package. To perform the tests, the selected player is *ExoPlayer*, provided developed by Google. This player supports DASH playback with Widevine CDM protection. If the media is protected and includes the Widevine Modular PSSH information, the player will engage all the protection and play rights issued in the license. Some Android devices also include support for PlayReady, in that case ExoPlayer can also use PlayReady's CDM.
- **iOS:** Apple was already supporting AES-CBC with *cbcs* encryption previous to the CMAF specification. Firstly, HLS only supported the MPEG-2 TS container which had issues with newer codecs, with that issues Apple decided to start supporting fMP4 on its protocol and devices. Starting from iOS 10.0, they supported fMP4 in HLS while protecting the content with FairPlay.

3.3.3. Others

There are multiple devices on the market that support streaming, and are important for VoD companies because customers owns them.

- **Google Chromecast:** this device was designed to work with old televisions that did not benefit from smart features. The device acts like a receptor and can be used as a streaming client for VoD services. It supports multiple protection systems, RakutenTV uses PlayReady, but it is also possible to use Widevine Modular. Google published in the Widvine news page that Chromecast was updated in summer 2018, the update included a new Widevine CDM with support for CMAF standard.
- **SmartTV:** in Europe most of the new televisions being sold are SmartTV which offer more features compared to the traditional televisions. VoD services usually make their application available in such devices, manufacturers such as Samsung, LG, Philips and Sony are key devices for RakutenTV.

  - **Samsung Tizen:** Samsung already supports DASH using protected fMP4 on their devices. The problem comes when CMAF requires AES-CBC on the encrypted medias. Samsung has not integrated the latest PlayReady Porting Kit, so SmartTVs still do not have any support for CMAF required encryption. It is expected to be included in the following years.
  - **LG WebOS:** LG is in a similar situation like Samsung. They do not currently support AES-CBC with the included DRM (PlayReady). No possibility either to currently support such device.
  - **Android TV:** some device manufacturers like Sony and Philips have used AndroidTV instead of building their own operating system. AndroidTV has the Widevine Modular CDM embedded in the operating system, so if the TV uses a version higher than 7.1, it will support AES-CBC with *cbcs* encryption.

- **Video Game Console:** over time, these devices are getting relevant, they are present in most of the living rooms of customer, someone who owns a gaming console is used to pay for a service, this is an interesting customer for a VoD service.

  - **Sony PlayStation 4:** currently it supports PlayReady for protected content, but the main problem is that uses an old Device Porting Kit that does not support AES-CBC. Next generation of PlayStation is to be released in 2020, so, no further updates in the PlayReady CDM are expected. Probably, this device will never support CMAF.
  - **Microsoft Xbox One:** in this case, Microsoft took a different approach and decided to add AES-CBC support to the Xbox One, their idea is to make the Xbox One the test-bed for all the PlayReady related tests. In this case, companies can use this gaming console to test CMAF encrypted medias with the PlayReady License server.

  They contain old versions of the PlayReady Porting Kit, and no further updates are expected on the included CDM, because new versions of this gaming consoles will be presented in 2020. They will not support CMAF content.

In general the device status confirms that CMAF is not mature nor production ready, where TV vendors are not including support for AES-CBC right now. It is estimated that starting from 2020, CMAF will start taking more relevance. Probably, if the mobile industry is moving towards CMAF, this will make device manufacturers to include support for the new encryption algorithm.

3.4. Conclusions

There are several open-source tools that can already generate CMAF packages, with the underlying fMP4 container and encrypting the content using AES-CBC with the *cbcs* scheme. Bento4 and Shaka Packager will be used to generate sample packages in the following section. Both packages should support signaling the appropriate DRM information on the streaming manifest, both for DASH and HLS.

Most common Digital Rights Management systems should support the CMAF content key delivery. In fact, they should support the delivery independent of the encryption mode, key lengths are equal in AES-CTR and AES-CBC. It is strange that PlayReady requires to signal the decryption type when returning the content key, the player could read itself the information on the manifest or in the appropriate box in the fMP4 and configure the decryption mode accordingly.

Finally, it seems that most of the connected devices such as smart tvs and gaming consoles do not currently support CMAF because they do not have the appropriate CDM and no further updates are expected. In case of Android and iOS it is expected to be able to play CMAF packages out of the box, they should be supported in the new versions of the operating system. Browsers should also support CMAF either because the CDM can be updated at the application level (Chrome), or the CDM is implemented by the operating system (Safari and Edge).

With all of these, in the next chapter the validation of the packagers, DRM systems and players will be carried out.

CHAPTER 4. IMPLEMENTING CMAF

This chapter will summarize how the analysis was carried out, and which devices successfully played the generated CMAF packages.

4.1. Development process

The implementation process during this chapter is shown on Figure 4.1, it mimics the steps done in RakutenTV's playback chain.

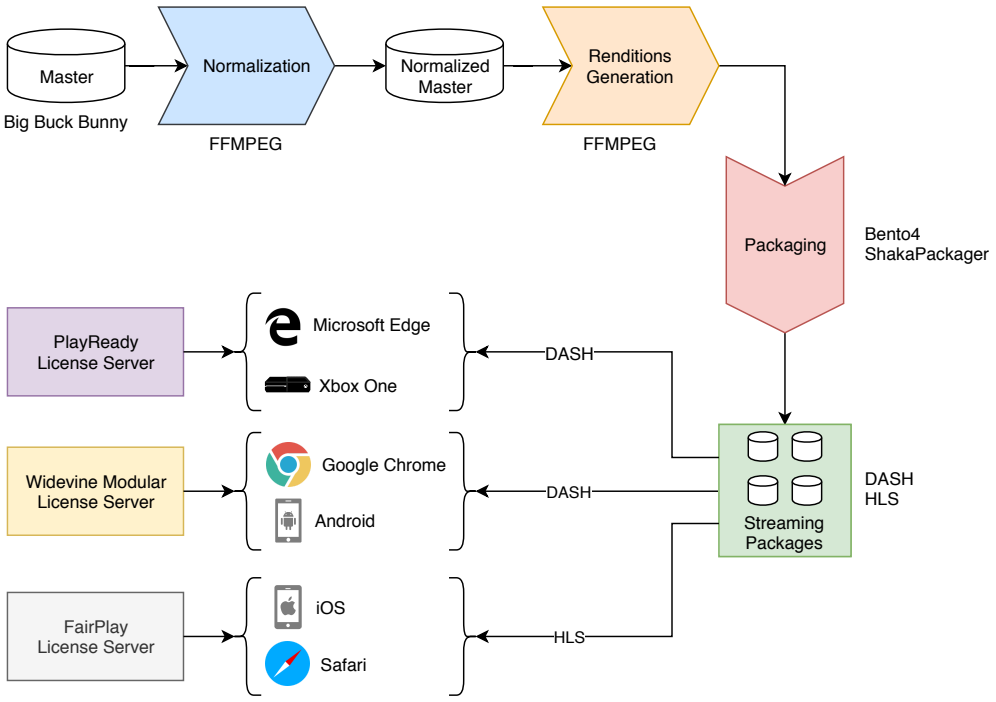

Figure 4.1: Process for CMAF packaging and device validation.

The following enumeration summarizes step by step (following Figure 4.1) the different stages on the execution of CMAF implementation:

1. **Master normalization:** the first step that would mimic RakutenTV process is the ingestion and normalization of a master video file. The input video will be a generic video available in internet with some random characteristics in terms of codecs and containers. The output of this process will be a normalized video where the container will use the MP4 format and the video stream will be encoded using the AVC codec (H.264). This step would make sure that the input file is not corrupt and at the same time make the output file to be compatible with the next steps. *Ffmpeg* is the encoding tool chosen to perform this job, it is a well-known open-source project used in most of the video and audio encoding engines.

2. **Renditions generation:** once the normalized master is available and follows the expected parameters it is time to generate the renditions. To generate them *ffmpeg* will

be used, using the normalized master will be mapped into several sub-quality output files, the encoding will transform the input video in multiple versions which different resolutions and bitrates, they will be used on the packaging step.

3. **Packaging:** the renditions will be encrypted and packaged into a streaming package using the CMAF specification. Bento4 and Shaka Packager will be used in this step, both of them support the encryption algorithms required in CMAF. The output package will contain the streaming manifests for DASH and HLS. Those manifests will include the necessary information that DRM systems requires to acquire the content key and engage the playback session.

4. **Storage and delivery:** this part will be different compared to RakutenTV, instead of using a distributed file system and delivering the content through different CDNs, the packages will be stored in the same encoding server. An HTTP file server will be used to serve the packaged content, the minimum requirement is that the HTTP server must support secure connections (HTTPS). Browsers require HTTPS to engage the Encrypted Media Extensions, so the content key is fetched securely, if HTTPS is not enabled the CDM will reject the license server.

5. **License Acquisition:** security is a fundamental part of this thesis and the execution of the test could not be properly done without using industry proven DRM solutions. Internal services have been adapted to support license delivery for CMAF packages. The existing PlayReady license server was upgraded with the latest version of the SDK adding support for AES-CBC key acquisition. Widevine Modular's server was not necessary to modify, the encryption scheme is not necessary when generating Widevine licenses. Finally, a FairPlay license server was implemented using the Golang project explained in the previous chapter, it was developed only the bare minimum logic to issue valid licenses.

6. **Device testing:** the last step is to check device compatibility with the generated packages, both DASH and HLS will be tested using the medias that are encryted with the *cbcs* encryption scheme. The testing will consist on a playback of the package, check if license is delivered and the playback session is properly started. Basically, if the device can play the new created package. Bento4 and Shaka Packager output manifests will be tested to verify that both options are suitable to be used on further implementations.

All the development has been carried out in an Ubuntu Server with enough CPU power to encode and encrypt flawlessly the master and its renditions. Docker has been used to simplify all the setup process required to get the encoding and packaging tools working. A Docker image has been created which contains all the software required to create CMAF packages. Check appendix B. *CMAF-Tools Docker Image* to know more about Docker and the *cmaf-tools* image built for this project.

4.2. Encoding

Everything starts with a sample master, the tests in this thesis were carried out using only a video stream, the audio was left aside (this simplifies the whole testing). The objective is to verify the viability of the project, so its better to keep things simple and controlled.

To perform the proof of concept, the selected media was *Big Buck Bunny*. It is an open-sourced short done with Blender. Since it was created, it has become a standard testing video on the industry.

Figure 4.2: Big Buck Bunny Cover. Video sample used on the thesis.

The listing 4.1 shows the output of *mediainfo* command when applied to the Big Buck Bunny sample file. Mediainfo command can be used to obtain the information of the media, including the codecs, container, duration, resolution, bitrate, frame rate, and other technical information of the stream.

The master video that has been used is encoded using MPEG-4 and stored in an AVI (Audio Video Interleave) container. The bitrate of the stream is variable, with an average of 9387 kbps and encoded at 24 fps (frames per second).

Listing 4.1: Command: mediainfo output of the original video track.

```
$ docker run --rm -v /root/master-thesis/:/media -w /media gerardsoleca/cmaf-tools mediainfo ↩
    big_buck_bunny_1080p_stereo.avi

General
Complete name                            : big_buck_bunny_1080p_stereo.avi
Format                                   : AVI
Format/Info                              : Audio Video Interleave
File size                                : 682 MiB
Duration                                 : 9 min 56 s
Overall bit rate mode                    : Variable
Overall bit rate                         : 9 587 kb/s
Writing application                      : MEncoder 2:1.0~rc2-0ubuntu13
Writing library                          : MPlayer

Video
ID                                       : 0
Format                                   : MPEG-4 Visual
Codec ID                                 : MP42
Codec ID/Info                            : Microsoft MPEG-4 v2 (pre-standard)
Codec ID/Hint                            : Microsoft
Duration                                 : 9 min 56 s
Bit rate                                 : 9 328 kb/s
Width                                    : 1 920 pixels
Height                                   : 1 080 pixels
Display aspect ratio                     : 16:9
Frame rate                               : 24.000 FPS
Compression mode                         : Lossy
Bits/(Pixel*Frame)                       : 0.187
Stream size                              : 663 MiB (97%)
```

From this source video file, the next step would be to normalize it. Treating media files in the same way is better to avoid issues on further steps. Things that will be normalized are the container and the codec with its own technical specifications.

Doing the normalization may prevent to perform costly encoding steps if the sample file is corrupt for example.

4.2.1. Master normalization

In this step, the master will be normalized according to different parameters, this would be the file that from which the different renditions will be generated. This normalized master will be encoded using the H.264 codec, and the output stored in an MP4 container, the resolution will be the same as the original master, which is in fullhd. The bitrate will vary depending on the content with a maximum of 15 mbps. Taking into account that the codec H.264 is more efficient compared to MPEG4, this will make that the resulting normalized master be smaller in size compared to the original master..

Listing 4.2 shows the ffmpeg command used to generate this intermediate file. It selects only the video stream from the reference file, and applies the H264 video codec. The parameters used on ffmpeg should generate a video stream with constant framerate, and a visually loss-less image. The maximum bitrate of the encoded file can be up to 15 mbps, but, because of the original master has an average bitrate of 9328 kbps, it is not expected to be over the value of the original master.

Listing 4.2: Intermediate video stream generated using ffmpeg and libx264 to envide the video with H264 codec.

```
$ docker run --rm -v /root/master-thesis/:/media -w /media gerardsoleca/cmaf-tools \
    ffmpeg -y -err_detect explode -xerror \
    -i big_buck_bunny_1080p_stereo.avi \
    -c:v libx264 -preset medium -maxrate 15m -bufsize 30m -crf 17 \
    -threads 0 -pix_fmt yuv420p -an \
    master-bbb.mp4
```

Listing 4.3 shows the mediainfo output for the intermediate file, it is possible to see that the codec being used is H.264, and it is stored in an MP4 container. Video bitrate has been reduced to 8123 kbps, which has a direct impact on the output stream size, the normalized master is smaller than the original one.

Listing 4.3: Mediainfo output for the normalized master. It shows the encoding and technical properties of the video stream.

```
$ docker run --rm -v $(pwd):/media -w /media gerardsoleca/cmaf-tools mediainfo master-bbb.mp4

General
Complete name                            : master-bbb.mp4
Format                                   : MPEG-4
Format profile                           : Base Media
Codec ID                                 : isom (isom/iso2/avc1/mp41)
File size                                : 578 MiB
Duration                                 : 9 min 56 s
Overall bit rate                         : 8 125 kb/s
Encoded date                             : UTC 1904-01-01 00:00:00
Tagged date                              : UTC 1904-01-01 00:00:00
Writing application                      : Lavf57.56.101
```

```
Video
ID                                       : 1
Format                                   : AVC
Format/Info                              : Advanced Video Codec
Format profile                           : High@L4
Format settings, CABAC                   : Yes
Format settings, ReFrames                : 4 frames
Codec ID                                 : avc1
Codec ID/Info                            : Advanced Video Coding
Duration                                 : 9 min 56 s
Bit rate                                 : 8 123 kb/s
Width                                    : 1 920 pixels
Height                                   : 1 080 pixels
Display aspect ratio                     : 16:9
Frame rate mode                          : Constant
Frame rate                               : 24.000 FPS
Color space                              : YUV
Chroma subsampling                       : 4:2:0
Bit depth                                : 8 bits
Scan type                                : Progressive
Bits/(Pixel*Frame)                       : 0.163
Stream size                              : 578 MiB (100%)
Writing library                          : x264 core 148 r2748 97eaef2
```

4.2.2. Generating adaptive streaming renditions

Once the normalized master has been encoded, the next step is to generate all the streaming renditions. On adaptive streaming, several video files are generated with multiple bitrates and multiple resolutions. Having multiple versions of the same video can help players to switch between video streams. Depending on the network quality, screen resolution and other parameters such as DRM License configuration, some resolutions or bitrates can be better than others when playing, a good streaming protocol will adapt the playback session to the network or device conditions, trying to maximize the quality of the video.

There are multiple ways to select the appropriate renditions for a given content, current trends in VoD services is to use artificial intelligence to detect the best combinations of resolution and bitrate optimizing both size and quality. Maximizing the quality while keeping the size low helps on reducing the required budget to store the content.

These renditions usually are also called as rendition leaders, because they are encoded in a progressive way, where each rendition has better quality compared to the previous one. The idea is that the player can switch up or down if the network can cope with the bitrate. The appropriate leader will depend on the content, in this thesis three different renditions will be generated, one per each standard resolution: FullHD (1080p) with a bitrate of 3000 kbps, HD (720p) using 2200 kbps and finally a SD (480p) at 1400 kbps.

Listing 4.4 shows the command to be used when generating the different renditions. It is more optimal to generate all the renditions at once instead of generating them one by one. In this case the ffmpeg will decode the normalized master and re-encode each mapped stream to a sub-quality rendition.

Listing 4.4: Ffmpeg command that generates the multiple renditions to be used in this thesis.

```
ffmpeg -y  -err_detect explode -xerror -i master-bbb.mp4 \
    -filter_complex "[0:v]split=3[file_out_0][file_out_1][file_out_2];[file_out_0]setdar=w*sar/h↩
        ,drawtext=fontfile=./usr/share/fonts/truetype/dejavu/DejaVuSans.ttf:text='1920x1080-3000↩
```

```
        k':fontsize=36:fontcolor=white,split[file_out_0_0];[file_out_1]setdar=w*sar/h,drawtext=↩
        fontfile=./usr/share/fonts/truetype/dejavu/DejaVuSans.ttf:text='1280x720-2200k':fontsize↩
        =36:fontcolor=white,split[file_out_1_1];[file_out_2]setdar=w*sar/h,drawtext=fontfile=./↩
        usr/share/fonts/truetype/dejavu/DejaVuSans.ttf:text='854x480-1400k':fontsize=36:↩
        fontcolor=white,split[file_out_2_2]" \
    -map "[file_out_0_0]" -c:v libx264 -pix_fmt yuv420p -preset fast -tune film -profile:v main ↩
        -b:v 3000k -minrate 3000k -maxrate 3000k -bufsize 6000k -s:v:0 1920x1080 -sc_threshold 0↩
         -g 48 -keyint_min 24 -coder 1 -bf 3 -refs 4 -an -movflags +faststart -map_chapters -1 -↩
        avoid_negative_ts 1 -shortest -vsync 1 -f mp4 encoding/1920x1080-3000k.mp4 \
    -map "[file_out_1_1]" -c:v libx264 -pix_fmt yuv420p -preset fast -tune film -profile:v main ↩
        -b:v 2200k -minrate 2200k -maxrate 2200k -bufsize 4400k -s:v:0 1280x720 -sc_threshold 0 ↩
        -g 48 -keyint_min 24 -coder 1 -bf 3 -refs 4 -an -movflags +faststart -map_chapters -1 -↩
        avoid_negative_ts 1 -shortest -vsync 1 -f mp4 encoding/1280x720-2200k.mp4 \
    -map "[file_out_2_2]" -c:v libx264 -pix_fmt yuv420p -preset fast -tune film -profile:v main ↩
        -b:v 1400k -minrate 1400k -maxrate 1400k -bufsize 2800k -s:v:0 854x480 -sc_threshold 0 -↩
        g 48 -keyint_min 24 -coder 1 -bf 3 -refs 4 -an -movflags +faststart -map_chapters -1 -↩
        avoid_negative_ts 1 -shortest -vsync 1 -f mp4 encoding/854x480-1400k.mp4
```

The previous command takes the normalized master, and then it applies a filter complex which splits the video into three different sub-streams and maps each sub-stream to an encoder, the encoder applies the H.264 codec algorithm and burns the codec configuration in the stream (easy to know which stream is being played). When the encoding finishes, there will be three renditions with the specified qualities.

Adaptive streaming requires a video with constant bitrate, that is why the minimum and maximum bitrate configured in the codec property is the same. Frame rate is also required to be constant with closed GOPs. For adaptive streaming it is mandatory to have I-Frames every N-frames, the previous command will enforce an I-Frame every 24 frames (*keyint_min* parameter specifies that).

Figure 4.3 shows all the renditions overlapped between them. Players will scale between this three resolutions and bitrates depending on their screen-size and Internet performance.

Figure 4.3: Overlapped captures for the three generated renditions.

4.3. Packaging

Once the renditions are generated, the next step to be carried out is the packaging, in this step the renditions will be encrypted and information relative to the encryption and the content protection systems will be added in the container. Medias must be fragmented and encrypted using the *cbcs* scheme.

The finality of this thesis is to support HLS and DASH streaming protocols without replicating the underlying content. Bento4 and Shaka Packager are able to generate both streaming manifests using the same renditions.

As stated on previous sections, Common Encryption works by using an AES key and a key identifier, so the media is encrypted with the key and it is identified with the KID. Then, DRM systems will use the requested KID to return the appropriate decryption key.

On the Table 4.1 the KID and the Content Key are represented, both in hexadecimal and base64 formats. In the next commands this values will be used within the packager, so the media gets encrypted and properly identified.

	KID	**Content Key**
Hex	a1cb5c9393807ca023d9c8d1d7ec5837	2673e1b813f36f083cd825e9721b3d15
Base64	octck5OAfKAj2cjR1+xYNw==	JnPhuBPzbwg82CXpchs9FQ==

Table 4.1: KID (Content Id) and Content Key to be used in the test medias.

4.3.1. Bento4

The first software being tested is the Bento4, which is the current packager used in the company. Reading its documentation, it clearly states that it will generate CMAF compatible medias, encrypting the content with AES-CBC using pattern encryption and properly dealing with the DRM information which must be added on the manifests. Bento4 provides several high-level scripts that will simplify the whole packaging operation.

To properly generate a CMAF media, Bento4 requires two different executions. The first one will fragment the input medias, generating fragmented fMP4 containers compatible with the standard. On the second step, the medias will be encrypted and the streaming manifests created.

4.3.1.1. Fragmentation

During the encoding process, the renditions have been generated using MP4 containers with constant frame rate and constant I-Frames, they are ready for the fragmentation process. The fragmenter requires an I-Frame to generate a new *mdat* box, containing the video data for a specified duration interval. Usually, the industry uses 2-second fragments. When streaming such content, the player will be able to switch between qualities every two seconds (although in a real case it would depend on the buffer length).

The following Bento4 command will fragment each rendition using a timescale of 10 MHz and a fragment duration of two seconds.

Listing 4.5: Bento4 to generate Fragmented MP4s (fMP4) from the encoded renditions

```
# Fragment the containers so it is suitable for streaming
docker run --rm -v $(pwd):/media -v /tmp:/tmp -w /media gerardsoleca/cmaf-tools \
  mp4fragment --index --fragment-duration 2000 --timescale 10000000 1920x1080-3000k.mp4 /tmp↩
      /1920x1080-3000k-frag.mp4

docker run --rm -v $(pwd):/media -v /tmp:/tmp -w /media gerardsoleca/cmaf-tools \
  mp4fragment --index --fragment-duration 2000 --timescale 10000000 1280x720-2200k.mp4 /tmp/1280↩
      x720-2200k-frag.mp4

docker run --rm -v $(pwd):/media -v /tmp:/tmp -w /media gerardsoleca/cmaf-tools \
  mp4fragment --index --fragment-duration 2000 --timescale 10000000 854x480-1400k.mp4 /tmp/854↩
      x480-1400k-frag.mp4
```

It is possible to compare the original and the fragmented MP4, The differences between them will be easier to notice. On the left side of the Figure 4.4 the normal MP4 container is shown, there is a single *mdat* and *moov* box. On the left, the fragmented one is listed, it has multiple *mdat* and *moof* boxes, they are repeated periodically, in this case each *mdat* will contain 2 seconds of video. The first fragment on each *mdat* box will be an I-Frame (full frame), so the player can successfully decode the portion of video contained in the fragment.

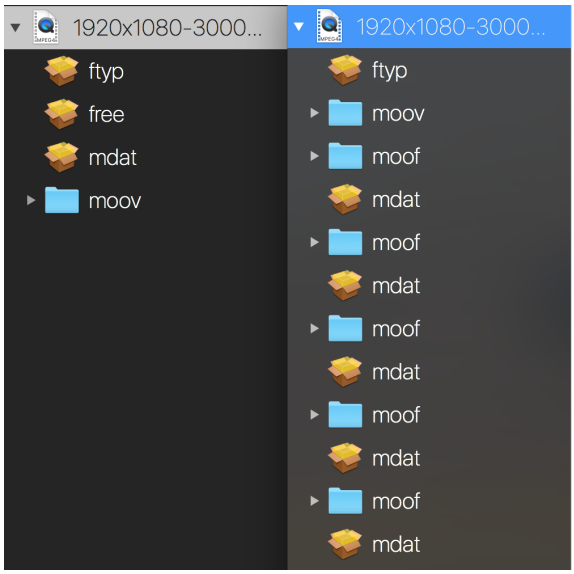

Figure 4.4: Boxes comparison between a MP4 and a fMP4.

4.3.1.2. Encryption and DRM signaling

After generating the fragmented containers, it is time to package the media. The packaging consists of encrypting the medias and then generate the streaming manifests for HLS and DASH. These manifests need to be properly signaled with the DRM information, so the players can securely fetch the content key and enable the playback on the device.

Bento4 offers a tool which encrypts, signals and generates the manifests. For a VoD service, the required DASH manifest is the *"on_demand"*, the encryption scheme for CMAF has to be set to *cbcs*. KID and Content Key are configured in the *encrypted-key* argument, both of them specified in hexadecimal. After configuring the encryption arguments, it is time to set the configuration for the DRM signaling. It is mandatory to provide the license

server url for PlayReady, and the configuration of the stream for Widevine. The generation of the PSSH data, which will signal the DRM is provided by Bento4. In the same command HLS is enabled, and by using the setting *hls-key-url* the FairPlay License server will be configured. Last but not least, all the renditions are added into the command and tagged as a video files.

Listing 4.6: Bento4: encrypt, signal DRM and generate HLS and DASH manifests with mp4dash.

```
docker run --rm -v $(pwd):/media -v /tmp/package/:/tmp/package/ -w "/media" gerardsoleca/cmaf-tools \
    mp4dash \
    -o bento4 --force --profiles=on-demand --mpd-name stream.mpd \
    --encryption-cenc-scheme=cbcs \
    --encryption-args="--global-option mpeg-cenc.piff-compatible:true" \
    --encryption-key a1cb5c9393807ca023d9c8d1d7ec5837:2673e1b813f36f083cd825e9721b3d15 \
    --playready-header=LA_URL:https://playready-license.server.com \
    --widevine-header=provider:cmaf-thesis#content_id:6131636235633933393338303763613032336439633864316437656335383337#protection_scheme:cbcs \
    --playready-add-pssh \
    --hls \
    --hls-key-url=https://fairplay-license.server.com \
    [type=video]/tmp/package/1920x1080-3000k-frag.mp4 \
    [type=video]/tmp/package/1280x720-2200k-frag.mp4 \
    [type=video]/tmp/package/854x480-1400k-frag.mp4
```

After running the command in the Listing 4.6 the created package will look like Listing 4.7. It contains the three encoded renditions, a *stream.mpd* which is the DASH manifest representation, and the *m3u8* files, that are the HLS manifests.

Listing 4.7: Bento4 package elements

```
-rw-r--r-- 1 root root       927 Jun  4 23:46 master.m3u8
-rw-r--r-- 1 root root 216473812 Jun  4 23:46 media-video-avc1-1.mp4
-rw-r--r-- 1 root root 158235350 Jun  4 23:46 media-video-avc1-2.mp4
-rw-r--r-- 1 root root 100924576 Jun  4 23:46 media-video-avc1-3.mp4
-rw-r--r-- 1 root root      3013 Jun  4 23:46 stream.mpd
-rw-r--r-- 1 root root     23750 Jun  4 23:46 video-avc1-1_iframes.m3u8
-rw-r--r-- 1 root root     23913 Jun  4 23:47 video-avc1-1.m3u8
-rw-r--r-- 1 root root     23661 Jun  4 23:46 video-avc1-2_iframes.m3u8
-rw-r--r-- 1 root root     23852 Jun  4 23:47 video-avc1-2.m3u8
-rw-r--r-- 1 root root     23444 Jun  4 23:46 video-avc1-3_iframes.m3u8
-rw-r--r-- 1 root root     23734 Jun  4 23:48 video-avc1-3.m3u8
```

4.3.2. Shaka Packager

Shaka Packager has a simplified command interface. It provides an all-in-one executable that will be in charge of the fragmentation, encryption, DRM signaling and manifest creation. The KID and content key will be the same as before.

The idea is the same, by default shaka-packager uses an *"on_demand"* manifest for DASH. When using the packager command, it expects the renditions as input files, and they need to be tagged using a *drm_label* (leave aside this, it is out of the scope of this thesis, this are parameters to be used on the license server to allow playback of certain tracks). Shaka Packager introduces 6 seconds in clear at the beginning of the stream, this leads to a non-encrypted media on the first seconds. Setting 0 to the *clear_lead* parameter enforces

that the encryption starts from the beginning. This packager by default tries to fetch the encryption key from Widevine or PlayReady Key Servers, that is why it is applied the raw encryption to be able to use specific KID and content key within the media. The other protection systems are Widevine, PlayReady and Fairplay, the packager will handle the PSSH generation

Listing 4.8 shows the issued command which will handle the fragmentation, encryption and manifest creation in Shaka Packager.

Listing 4.8: Shaka Packager: encrypt, signal DRM and generate HLS and DASH manifests with packager.

```
docker run --rm -it -v $(pwd):/media -w /media gerardsoleca/cmaf-tools \
  packager \
    in=1920x1080-3000k.mp4,stream=video,output=shakapackager/1080p.mp4,drm_label=HD \
    in=1280x720-2200k.mp4,stream=video,output=shakapackager/720p.mp4,drm_label=HD \
    in=854x480-1400k.mp4,stream=video,output=shakapackager/480p.mp4,drm_label=SD \
  --protection_scheme cbcs \
  --clear_lead 0 \
  --enable_raw_key_encryption \
  --keys label=SD:key_id=a1cb5c9393807ca023d9c8d1d7ec5837:key=2673e1b813f36f083cd825e9721b3d15,↩
      label=HD:key_id=a1cb5c9393807ca023d9c8d1d7ec5837:key=2673e1b813f36f083cd825e9721b3d15 \
  --protection_systems CommonSystem,Widevine,PlayReady,FairPlay \
  --mpd_output shakapackager/stream.mpd \
  --hls_master_playlist_output shakapackager/stream_master.m3u8 \
  --hls_key_uri https://fairplay-license.server.com
```

The generated package contains all the encrypted renditions and the manifests to play both HLS and DASH streams. Shaka Packager's output is shown in the following listing:

Listing 4.9: Shaka Packager package elements

```
-rw-r--r-- 1 root root 216419817 Jun  4 20:42 1080p.mp4
-rw-r--r-- 1 root root 100870597 Jun  4 20:42 480p.mp4
-rw-r--r-- 1 root root 158181355 Jun  4 20:42 720p.mp4
-rw-r--r-- 1 root root      5545 Jun  4 20:42 stream_0.m3u8
-rw-r--r-- 1 root root      5646 Jun  4 20:42 stream_1.m3u8
-rw-r--r-- 1 root root      5502 Jun  4 20:42 stream_2.m3u8
-rw-r--r-- 1 root root       446 Jun  4 20:42 stream_master.m3u8
-rw-r--r-- 1 root root      3237 Jun  4 20:42 stream.mpd
```

4.4. DRM and Licensing

Both packagers inject similar DRM information in the streaming manifests. The different DRM systems define how the medias must be signaled, so the embedded CDM in the device can use such information to acquire the content key using the license server.

Below are shown how the different DRM information is written in the DASH or HLS manifest:

4.4.1. Widevine Modular

In this thesis Widevine Modular is signaled in the DASH manifest, so Android and Chrome can properly use their CDM. This DRM system is identified by a unique uuid: *edef8ba9-*

79d6-4ace-a3c8-27dcd51d21ed, which players use to fetch the DRM information and set up the CDM.

Listing 4.10 shows how the Widevine Modular is specified in the DASH manifest, it uses a *ContentProtection* object identified with the pertinent uuid. The contained data is a binary payload in base64 format. That payload will contain the VoD provider who licenses this content and the KID among other proprietary data to be used by the CDM.

Listing 4.10: Widevine Modular definition in a DASH manifest.

```
<!-- Widevine -->
    <ContentProtection schemeIdUri="urn:uuid:edef8ba9-79d6-4ace-a3c8-27dcd51d21ed">
      <cenc:pssh>AAAAXXBzc2gAAAAA7e+LqXnWSs6jyCfc1R0h7QAAAD0IARIQoctck5OAfKAj2cjR1+↩
          xYNxoFd3Vha2kiIGExY2I1YzkzOTM4MDdjYTAyM2Q5YzhkMWQ3ZWM1ODM3</cenc:pssh>
    </ContentProtection>
```

4.4.2. PlayReady

In this case, PlayReady is only supported in DASH, and as Widevine, the DRM data is stored in a *ContentProtection* object inside the manifest. The player will use the system identifier *9a04f079-9840-4286-ab92-e65be0885f95* to obtain the PlayReady Protection Header and make the CDM request the information with it.

Listing 4.11: Shaka Packager package elements

```
    <!-- PlayReady -->
    <ContentProtection schemeIdUri="urn:uuid:9a04f079-9840-4286-ab92-e65be0885f95">
      <mspr:pro>↩
          XAIAAAEAAQBSAjwAVwBSAE0ASABFAEEARABFAFIAIAB4AG0AbABuAHMAPQAiAGgAdAB0AHAAOgAvAC8AcwB↩
          ...
      </mspr:pro>
    </ContentProtection>
```

The data inside the *"mspr:pro"* field is an XML encoded in base64, Listing 4.12 shows the decoded content. It is important to notice two things, the first one is that the encryption algorithm is not properly configured. The *ALGID* specifies that the request will be for an AES-CTR media, but the media has been encrypted using AES-CBC, this must be modified so the playback works properly when requesting the key. The second thing to notice is the KID, even it is different from the Listing 4.1 it is correct, PlayReady changes the endianess format of the KID so the encoded base64 changes.

Listing 4.12: Shaka Packager package elements

```
<WRMHEADER xmlns="http://schemas.microsoft.com/DRM/2007/03/PlayReadyHeader" version="4.0.0.0">
    <DATA>
        <PROTECTINFO>
            <KEYLEN>16</KEYLEN>
            <ALGID>AESCTR</ALGID>
        </PROTECTINFO>
        <KID>k1zLoYCToHwj2cjR1+xYNw==</KID>
        <CHECKSUM>vAVr98qLJdo=</CHECKSUM>
        <LA_URL>http://playready-license.server.com</LA_URL>
    </DATA>
</WRMHEADER>
```

4.4.3. FairPlay

FairPlay is only supported in HLS manifests because Apple's DRM does not support other streaming protocols. Listing 4.13 shows how the FairPlay information is added to the streaming manifest. The *"EXT-X-KEY"* is used to configure the player with the corresponding information. *SAMPLE-AES* is the code that Apple uses for AES-CBC using pattern encryption (*cbcs*), and the URI contains the URL of the DRM license server that the player will use to fetch the content key. Finally, the players will use the *"KEYFORMAT"* to identify which kind of DRM system must be used and then engage it.

Listing 4.13: FairPlay information embedded in the HLS manifest.

```
#EXT-X-KEY:METHOD=SAMPLE-AES,URI="https://fairplay-license.server.com/↩
    a1cb5c9393807ca023d9c8d1d7ec5837",KEYFORMATVERSIONS="1",KEYFORMAT="com.apple.↩
    streamingkeydelivery"
```

4.5. Playback Testing

Once medias were generated it was time for the playback testing. Shaka Player and DASH-IF were used to build a simple testing page, while ExoPlayer for Android was compiled and configured to fetch the generated manifests by both packagers. Below the results are listed showing which devices worked properly and which ones failed:

4.5.1. Browsers

Following items summarize the results carried out using the different browsers that are widely used:

- **Google Chrome:** as it was expected, current stable versions of Google Chrome support the new CMAF standard. Both DASH-IF and Shaka Player have been able to play the DASH manifests using Widevine Modular.

  - **Firefox:** this browser has a deal with Google were Widevine Modular support is given to Firefox. Firefox's CDM is exactly the same that is found in some versions of Chrome. The problem, in this case, was that the DASH package could not be played, the CDM that Firefox is using in the latest release does not support AES-CBC. It could be expected that in future releases of this browser the CDM gets updated with the newer algorithms.

- **Microsoft Edge:** Edge was tested using the DASH manifest of both packages, none of them worked. Bento4 and Shaka Packager did not signaled correctly the encryption algorithm on the protection header of the manifest. But once corrected, Edge was still failing to play the content. Supposedly, Microsoft confirmed in the PlayReady conference held in New York in 2018 the support of CMAF in Edge browser, but next years conference, in 2019, they said that the Windows development team was facing some issues when porting the CDM with AES-CBC support in

Windows. Because of this, there is still no support for CMAF packages in Microsoft's browser.

- **Apple Safari:** as it has been explained, Safary supports HLS directly in the browser, it recognizes the manifest and performs all the steps to acquire the content key and decrypt the files. It successfully played the media built with both packagers. No issues with fMP4 nor the *cbcs* encryption scheme. The unique problem was that for this thesis no production certificate was granted by Apple, so the tests where carried out using the clear key method. Even dough, Apple confirms that if the playback works using the clear key method, then, the same media will work when using FairPlay for license acquisition.

4.5.2. Smartphones

The playback results for both platforms are shown below:

- **Android:** ExoPlayer successfully played the DASH streaming protocol of the Bento4 and Shaka Packager. The DRM systems that it was used was Widevine Modular, so it is possible to confirm that CMAF is supported in the Android platform. Back on June 2018, the Bento4 package could not be played, so this clearly states that there is ongoing work to support CMAF both in Android and in the packager. The problem was how the media was being encrypted by Bento4. As it has been stated, nowadays, both packagers can build a working CMAF package for Android.
- **iOS:** this is the same case as Safari, the result of both packagers could be played in the Apple's phone using the HLS manifest. As it was expected, the newer iOS versions perfectly support the fMP4 container.

4.5.3. Others

As it was expected, most of the connected devices did not have an updated CDM which supported *cbcs* encryption scheme. PlayStation 4, SmartTVs from LG and Samsung did not work with the DASH version of the CMAF package.

Below are shown the test results for the expected working devices:

- **Android TV:** this is the same case as Android, if the operating system version is higher than 7.1 and they include the Widevine CDM, ExoPlayer will be able to handle the playback of DASH streams using the CMAF package.
- **Chromecast:** this device from Google was the first one getting support for AES-CBC in the Widevine CDM. Back on summer 2018 the playback test was successfully done with the DASH stream and using the Widevine Modular license server. This device is already compatible with CMAF packages.
- **Xbox One:** on 2019 PlayReady conference Microsoft shown examples of an Xbox One playing a CMAF package using DASH streaming. But unfortunately, the device could not be tested within this thesis because there where no units to test.

4.6. Conclusions

Although it seems that both packagers properly encrypt and generate the streaming manifest there are some side issues with the DRM signaling that should be solved in their source code.

The PlayReady Protection Header is not properly built for none of the packagers, if the protection header is decoded it shows that the media is encrypted using AES-CTR although it is using AES-CBC. This can be circumvented manually by modifying the header with the correct information.

HLS has also an issue with the DRM signaling, both packagers add DRM information for Widevine Modular and FairPlay, Safari and iOS could not properly play the HLS file until it was modified by removing the Widevine Modular information. On a production ready system, there should be a post-processing step where some sanitation is done in the manifest.

Finally, it seems that CMAF support is being slowly added into multiple devices, by now, it covers browsers and smartphones, but there is lack of support for connected devices such as SmartTVs which are key devices in RakutenTV. Most of the revenue comes LG and Samsung televisions and it is expected to have CMAF support on them starting on 2020 onward.

CHAPTER 5. OPERATIONAL BENEFITS

On this chapter the impact of CMAF in a VoD service such as RakutenTV will be carried out. It is important to understand the benefit of the new package, why makes sense to introduce it to the platform and finally which is its economic impact.

5.1. Streaming packages in RakutenTV

Currently, playback department at RakutenTV is managing two main packages for copyright content. The first one is Widevine Classic, as it has been explained, it is a legacy Google technology supported by old devices. The second one, which is the main package in the company, supports both DASH and MSS sharing the underlying medias.

It is important to evaluate the technology, to know how flexible and optimized is and which problems could face in a long-running term. To contextualize the situation, lets imagine the following movie characteristics which will be the source of the different streaming packages generated in the company.

- **Video qualities:** the adaptive streaming package should contain resolutions for all screen sizes, if the master video has a FullHD resolution, the streaming packages will support the following resolutions:
  - FHD
  - HD
  - SD
- **Audio languages:** content is offered in multiple languages, so the customers can enjoy the cinema in the language they want. Typically, in the platform there are five main languages which are the following ones:
  - Spanish *(SPA)*
  - French *(FRA)*
  - Italian *(ITA)*
  - English *(ENG)*
  - German *(DEU)*
- **Audio qualities:** to offer a premium experience, the customer should be able to listen high-quality audio sound. The platform supports audio qualities from stereo to surround:
  - Stereo *(2.0)*
  - Surround *(5.1)*
  - DTS core *(dtsc)*

With this information, the following two sections will briefly introduce the streaming package map for RakutenTV.

5.1.1. Widevine Classic

Figure 5.1 shows all the packages that will be generated following the previous premises. Widevine Classic does not support multi-resolution streaming, so each package can only contain renditions using the same resolution. Due to this, only packages for SD and HD

resolutions are generated, FHD with Widevine Classic is not generated by two reasons: cost of the package and playback experience for the customer. Widevine Classic is not multi-resolution, if a user uses a FHD package in Widevine Classic it will not be able to switch to lower qualities, this will end up with buffering and a bad experience for the customer.

Another constrain for Widevine Classic is that each package can only contain one language, no multi-language packages can be built with this technology. This forces to duplicate the video stream per each audio, to minimize the cost of the whole package only stereo audio qualities are added.

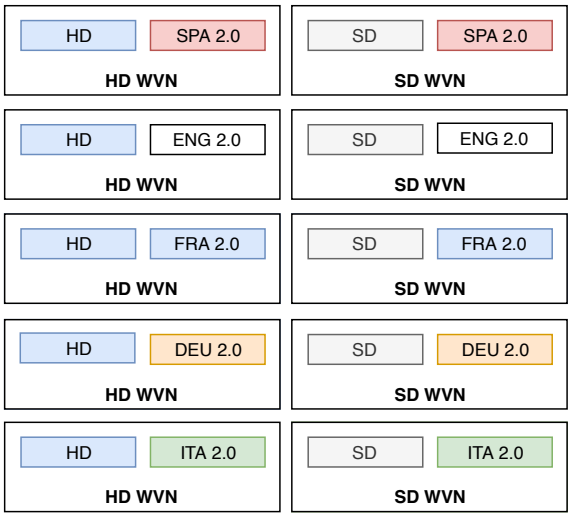

Figure 5.1: Widevine Classic package, each package contains one video resolution and one audio language.

Each box in the previous image represents a Widevine Classic package. Each package will be a video stream with a single audio language. So, the example movie that will be packaged will require ten Widevine Classic packages, two per each audio language. As it is possible to foresee, this will not scale if the amount of languages grow, storage costs multiply per each added language.

5.1.2. DASH and MSS

DASH and MSS packages are widely supported in almost all devices that can be used within the platform. As explained before, DASH and MSS support CENC encryption with multiple renditions and multiple audios, from a technical point of view both protocols share the same underlying medias, this makes a great advantage compared to Widevine Classic, two protocols are supported in the same package. Moreover, this package offers higher qualities compared to the previous one, it can handle higher resolutions and even multiple audio qualities and audio languages, all stored in separate files inside the same package.

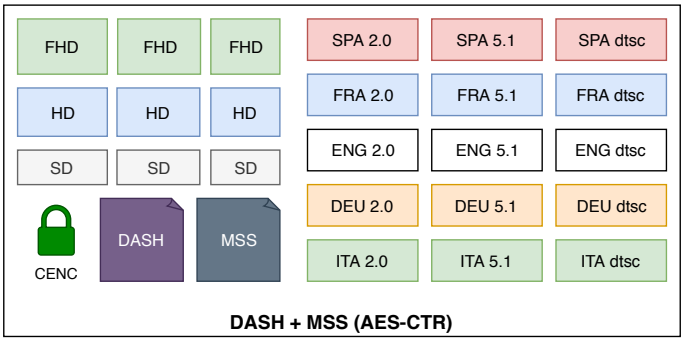

Figure 5.2: Current DASH and MSS package with multiple audio languages and audio qualities.

The previous figure represents the contents of DASH and MSS package, the two streaming protocol manifests will contain references to all the video and audio files. The Streaming manifest will hold all the video renditions with its information, all the languages and its qualities. Each language is stored separately from the video file, so the player can choose the desired audio for the video. Having independent audio files from the video makes the system to scale better. Adding a new audio will not duplicate the whole video, this package is extremely flexible and efficient because the video, the different audios and the streaming manifests are decoupled.

5.2. Continental expansion

Since 2019, the company is facing a continental expansion where its presence has moved from 14 countries to 42. Offering a VoD service in 42 countries force RakutenTV to optimize as match as possible all the streaming chain including the package creation and operation. In the initial 14 countries there were five main languages: Spanish, English, French, German and Italian, but when moving to 42 countries, new languages must be added into the platform. The content ingestion team is in charge of adding new languages to the current content and the new films are being ingested with more than five languages. The number of languages supported in the platform are expected to grow from five to eleven in average.

Now the idea is to model the different package sizes to evaluate and understand how both of them performs. RakutenTV usually considers that a movie has an average duration of 120 minutes, and the master video file uses FHD resolution and contains five languages. With this information, the package sizes can be modeled in the following way:

- **Widevine Classic:** as it has been explained before, the amount of packages and the size of them will depend on the number of languages that are included in the master file. In Widevine Classic two package resolutions are built, one with the renditions for the HD resolutions and the other one with the SD resolutions. Each package will only contain one language, so for example, there will be one HD and one SD for the English language, for the Spanish there will be again two packages. In average, because the films are encoded using constant bitrate it is possible to consider an average size per package of: 7.5 GB for the HD (including a single language) and

3.6 GB for the SD (also including one language). The total size of the required packages for the master will be the sum of HD and SD package multiplied by the amount of languages.

- **DASH and MSS:** as it has been said, this two protocols use adaptive streaming supporting all the resolutions in the market. The master file will be encoded into multiple renditions ranging from the lower SD ones to the higher FHD. This kind of package includes separate files for video and audio languages (including its qualities), so with the current example, the package will contain all the resolutions files and 15 audio files (there are three audio quality files per audio language). In average, the size of a 120 minute package will be around 11.59 GB (the sum of all the renditions), while each language contributes to the package size with 1.08 GB (considering three audio qualities: 2.0, 5.1 and dtsc).

Figure 5.3 shows how the number of languages impact on the size growth of each package. If the movie has only one language the combination of Widevine Classic SD and HD packages is smaller compared to the DASH and MSS one. Although the adaptive package will contain higher resolutions and video qualities (better quality of experience). But when two languages are started to being used, DASH and MSS package size will be smaller than the combination of the two Widevine Classic packages. When moving to 11 languages it can be clearly seen that the adaptive package outperforms in terms of size, this has a direct reduction on the storage costs. Apart from this, DASH and MSS are more optimal when streamed through the CDN, the same video file is shared across different languages (so different countries will have the same video stream), this helps on keeping the content cached in the CDN because is being used by customers all around Europe. Having warm caches in the CDN reduces costs by avoiding content requests to the origin, and it also reduces the buffer ratio because the content is already in the CDN network.

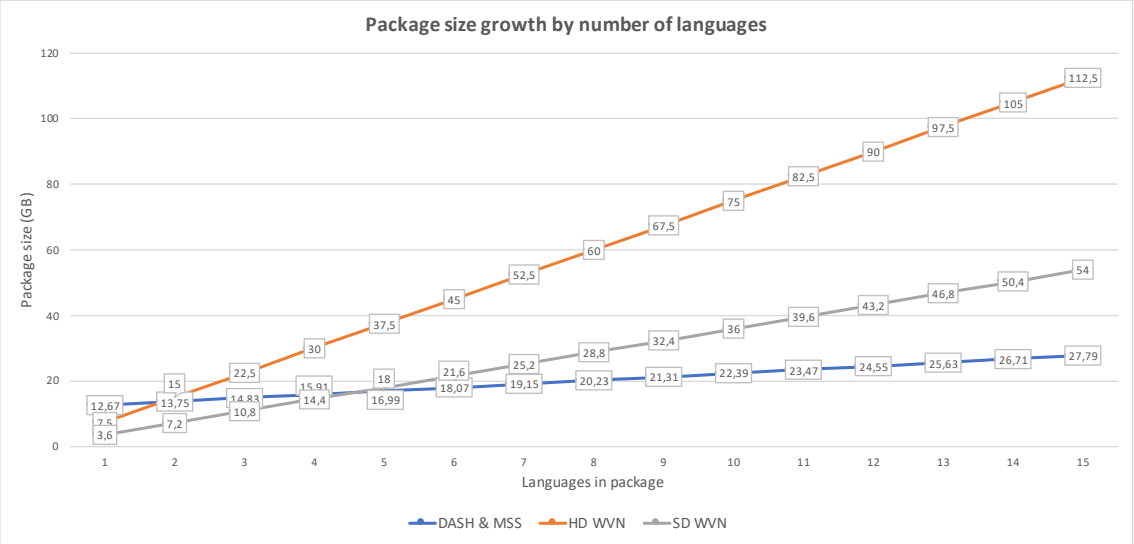

Figure 5.3: Graph showing the evolution of the Widevine Classic and DASH with MSS package size by the number of languages.

Another analysis that can be done, is the relative distribution of packages inside the platform, how does affect the evolution of streaming year over year. Although details of the

platform are confidential, Table 5.1 shows an approximation of how many masters could be ingested in the platform grouped by the included languages.

2017	
Movies	**Languages**
3156	1

2018	
Movies	**Languages**
3128	1
862	2
52	3
38	4
245	5
1	6
4	7

2019	
Movies	**Languages**
2228	1
733	2
74	3
23	4
64	5
21	6
11	7
5	8
4	9
4	10
6	11

Table 5.1: Ingested masters grouped by the amount of included languages and the year of ingestion.

Graph 5.4 shows the previous data in format of a relative distribution. On 2017 all the content ingested in the platform had a single language, because of this DASH and MSS package required more storage than the packages in Widevine Classic. When the ingested content started to have multiple audios the adaptive package started to outperform the other in terms of storage, around 55% of the total cost was due to the inefficiency of Widevine Classic packages, the remaining part was used by the adaptive streaming.

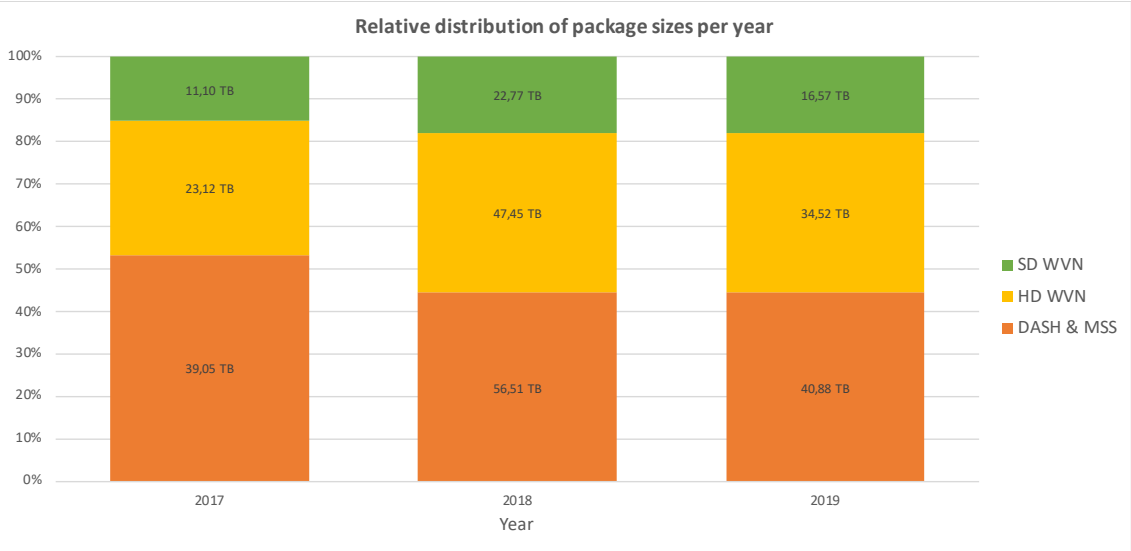

Figure 5.4: Graph that shows the relative distribution (in size) for all the streaming packages generated in a year.

From the previous graphic it can be seen that the content is not being efficiently ingested, there is still room to improve by increasing the number of languages per package. Usually,

content ingestion inserts the masters in pairs of local language and English language, improving this and having all the content in the same streaming package will enhance the metrics and reduce the operational costs. Less budget to keep the medias stored.

5.3. Moving forward with CMAF

In the previous section it has been presented the current package state in RakutenTV, none of them is suitable for Apple devices because they do not support DASH nor CENC encryption. The company did not generate the old MPEG-2 TS package for Apple devices because it was limited to such platform. The cost of maintaining all the medias using MPEG-2 TS will be similar to duplicating the cost of the current adaptive package, it does not make sense the required investment just for the Apple support.

With the new CMAF standard, the support is expected to be widely once television manufacturers start to include the updated CDMs on their devices. In the case of Google and Apple devices they already support CMAF, so building the package will, at least, work for these class of devices.

The unique difference between DASH and MSS package, and CMAF package is the encryption algorithm and the supported streaming protocol.

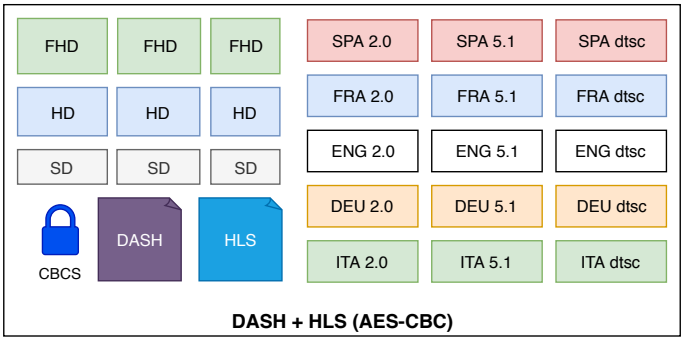

Figure 5.5: Current DASH and MSS package with multiple audio languages and audio qualities.

Figure 5.5 shows the new package where the encryption scheme changes and HLS is included, the audio languages (and their qualities) and the renditions are encrypted using the *cbcs* scheme. The packaging step that already packages the DASH/MSS package

NO ESTA ACABAT

5.4. Estimation of costs

Knowing that the playback department wants to deprecate Widevine Classic packages, it is possible to estimate the feasibility of using the cost of Widevine Classic to build and maintain the CMAF package.

Widevine Classic packages are being maintained for the LG TVs because on the past several problems with the MSS protocol were detected. There is an ongoing effort to

move the streaming protocol from Widevine Classic to MSS in these televisions. Once successfully changed, the legacy packages could be finally deprecated and removed from the platform.

5.4.1. Average cost per package

The playback team is in charge of developing and maintaining the encoding servers that perform the encoding and packaging. It is out of the thesis context, but the encoding platform is based on Amazon Web Services (AWS) and the software has been written to scale according to the encoding and packaging queue. Figure 5.6 shows a simplified view of the encoding platform, it is based on an orchestrator that manages which jobs are to be processed and which ones are already done. Finally, all the process uses information from the database.

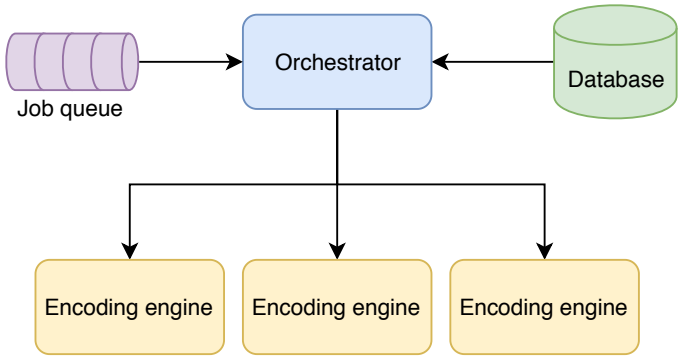

Figure 5.6: Simplified architecture for the encoding and packaging platform.

The previous simplified view runs on AWS instances, using EC2 servers for the orchestrator and the encoding engines, the job queue is based on ElastiCache and the database is an RDS (based in MySQL).

Then, all the medias are stored in an S3 bucket also in AWS which serves as the origin for RakutenTV CDNs. Table 5.2 summarizes the operation cost of the encoding platform using the public prices from the AWS page.

Operational Costs		
Operation		**Cost**
Encoding platform		0.592 €/hour
S3	Storage	0.022 €/GB (monthly basis)
	Transfer	0.085 €/GB (monthly basis)

Table 5.2: Summary of the operational cost only by infrastructure and storage.

It is possible to estimate the required times for encoding and packaging taking into account the packages explained before. Considering a 120 minutes FHD movie with five languages and three different audio languages per quality, the required time to encode all the renditions and audios will be around 3 hours. In 3 hours the platform generates all the renditions, the ones to be used in the DASH and MSS package, and the ones that are using within the Widevine Classic package. The next step is the encryption plus packaging,

as it has been seen in the platform, packaging an SD Widevine Classic tooks around 6 minutes per language while the HD versión only requires 3 minutes per language. The HD version contains less renditions than the SD Widevine Classic package, due to this, the required time is lower. For DASH and MSS, the time increases to 12 minutes per a package containing all the language files (five languages with three qualities per language), in this case, adding a new language only adds around 30 seconds to the estimated time. Finally, the CMAF package will be similar to the DASH and MSS package, it uses the same underlying medias and it only changes the encryption scheme, this would be again 12 minutes.

Operational Time		
Operation		**Time**
Encoding		3 hours
Packaging	SD WVN	6 minutes · language
	HD WVN	3 minutes · language
	DASH and MSS	12 min
	CMAF	12 min

Table 5.3: Summary of the time required to encode all the renditions and generate each package with the required renditions.

With the price information from Table 5.2, the processing times from Table 5.3 and the size of each package (previous section), it is possible to estimate the cost of each package taking into a count that the encoding and package step is carried out once and the storage and transfer costs are in a monthly basis.

- **Cost for Widevine Classic, DASH and MSS:**
  - **SD WVN:** 0.296 €(packaging) + 1.902 €/month
  - **HD WVN:** 0.148 €(packaging) + 4.013 €/month
  - **DASH and MSS:** 0.188 €(packaging) + 1.182 €/month
  - **Total:** 1.776 €(encoding) + 0.632 €(packaging) + 7.097 €/month (storage)
- **Cost for DASH and MSS, CMAF:**
  - **DASH and MSS:** 0.188 €(packaging) + 1.182 €/month
  - **CMAF:** 0.188 €(packaging) + 1.182 €/month
  - **Total:** 1.5 €(encoding avoiding WVC renditions) + 0.376 €(packaging) + 2.364 €/month (storage)

From the previous details it can be deduced that the costs of encoding will change a little because the Widevine Classic renditions could be avoided, less videos to encode. The packaging could be reduced by a 50% when using only DASH and MSS along with the new CMAF package. Having separated files requires less packaging time compared to Widevine Classic where each language requires a full combination of audio and video which takes more time to encrypt. Related to this, the storage and transfer costs are three times lower with the combination of CMAF and current adaptive package compared to the

Widevine Classic with the adaptive package. This difference of price could make feasible the introduction of CMAF in the company operations when deprecating Widevine Classic.

Take into account that this example is only carried out for a movie in a single month, expanding this to a thousand movies plus a similar number of episodes for shows, and month over month of catalog will increase the total bill of operations. Having more efficient packages reduce the overall required budget and improves company KPIs related to the quality of streaming.

5.4.2. Repackage

Finally, in order to build the CMAF packages for the old medias it would be possible to perform a repackage instead of doing full encoding from the master and then the package with the new encryption.

A repackage consists of decrypting the medias of the current DASH and MSS package, and encrypting them again using the *cbcs* scheme. The source files (previous to the encryption) are already valid for CMAF, this avoids the cost of re-encoding the whole catalog just for building the new CMAF packages. Packaging would take up to an hour, 30 minutes for the decryption and 30 minutes for the encryption, making the process efficient and relatively low cost.

5.5. Conclusions

RakutenTV principal streaming mechanism is Dynamic Adaptive Streaming over HTTP and Microsoft Smooth Streaming because it is supported by the almost all the devices in the platform. Both solutions share the same underlying medias which are encrypted using AES-CTR with full encryption (*cbcs* scheme). With this solution, streaming on Apple devices is not completely supported. The new standard pursued by the OTT companies in the streaming market supports Apple devices, and most of the new Android and Chrome versions are supported too. If the Widevine Classic package is finally deprecated there will be enough budget to repackage all the medias to CMAF, and start packaging the new medias also with CMAF (apart from the DASH with MSS pacakge).

As far as Widevine Classic is deprecated, CMAF can be maintained with less than the required budget for the legacy protocol.

Then, having two adaptive streaming packages in the platform could ensure good KPIs for Apple devices (including iOS and Safari), and the rest of the devices. Having efficient medias helps on reducing the required operational budget, and increases the efficiency of the CDN keeping the content more warm.

CHAPTER 6. CONCLUSIONS

6.1. Project Conclusions

CMAF is becoming more mature year over year, when this thesis started back in 2018 there was no support in Chrome and support for Apple was recently added. By that time, Microsoft's license server already allowed AES-CBC, but Xbox One was the unique device supporting such encryption method.

One year later it has been proven that VoD providers are moving towards CMAF, support is increasing over time. Even Microsoft presented a full encoding and streaming platform based on their Azure cloud, their service generates DASH and HLS packages using the old encryption scheme (*cenc*) and the new one (*cbcs*). Back in 2018, it was expected to see SmartTVs supporting AES-CBC with *cbcs* starting from 2020. Currently, there is no public commitment from them to add support for the new packages. Probably, they will be forced to support such content once Netflix, HBO or big service providers start requiring it to be present on the devices.

Although Bento4 and Shaka Packager support *cbcs* encryption and it has been proven to work during this thesis, they require some modifications to properly support all the DRM combinations or even fix the issue with the PlayReady Protection Header. Shaka Packager gives a better feeling compared to Bento4, the packaging process is quicker and involves less steps. Apart from this, Shaka Packager is developed by Google which, in theory, has more strength to keep it up to date.

With all the knowledge acquired developing the thesis, it can be confirmed that in a real production environment no new renditions will be required. Avoiding new renditions for CMAF helps on taking advantage of the already encoded renditions for DASH and MSS, this has a direct impact in cost savings (no extra encoding time is required). The packaging time for adaptive streaming can be despised compared to Widevine Classic, new files only contribute to some seconds of encryption, instead, the Widevine Classic requires to encrypt every time the video stream which is a lot of data and requires more time.

The cost of CMAF can be simplified to the cost of the storage, the encoded medias will be shared with the current CENC (DASH and MSS) package and the new CMAF package, because of this, the encoding and packaging price is not relevant when thinking about adding this new package to the platform.

There are some benefits behind CMAF that could make the company willing to implement it:

- Apple's iOS devices will be properly supported without requiring custom players, or expensive third party solutions which do not actually provide good performance. RakutenTV has developed a custom player which handles DASH streams with AES-CTR and it is able to decrypt and transmux on the fly the content to AES-CBC. The development has been carried out in an entire year by an iOS engineer, all the knowledge of the system is only held by that engineer. This is a problem for the company, where no other employees know how do the system work. This have some side issues like maintenance, development time and upgrades on the software, one engineers cannot cope with all the work that developing a player implies.

- The in-house solution only works for iOS, but Safari or AppleTV are out of the equation, they are not supported in RakutenTV. With the introduction of CMAF, these devices could be supported properly and customers will be satisfied. Company's customer service usually explains that most of the customers requests support for Safari and AppleTV, supporting these devices could have a direct impact on revenues.

On the operational benefits chapter it has been shown that deprecating Widevine Classic would reduce the operational cost of the company. There is an ongoing effort to make this happen on 2020, were the remaining devices that are using this legacy system will be removed from the platform or upgraded to newer protocols such as Smooth Streaming. If the deprecation finally happen there would be enough budget to start supporting the new CMAF format. It is more efficient compared to Widevine Classic, and requires less budget to be maintained. This would help on the company expenses, both from a technical perspective and from the human management; ingestion and operation teams need to take care of less packages and combinations which at the end has an impact on the operational costs.

Finally, having efficient streaming packages such as CMAF or the current CENC where multiple language files are under the same package helps on keeping a good cache hit ratio in the CDN. If all the audios share the same video all the countries will have at least a cached copy of the video stream. This helps reducing, again, the operating costs of the platform, if the content is well cached in the CDN there is no requirement to fetch the content from the origin. The origin traffic is much more expensive than the CDN traffic, because of this, it is important to keep all the content well cached in the CDN.

6.2. Achieved Objectives

The objectives for this thesis where listed in the project overview chapter, below the outcome of the study has been summarized:

- **CMAF pakcages:**
  - **Streaming packages:** the reference video has been packaged and tested successfully.
  - **Use AES-CBC:** *cbcs* is supported both by the packagers and some of the players.
  - **Use content protection systems:** all of them work except PlayReady. Edge did not support *cbcs*. Xbox One could not be tested, so there is no granted support confirmed within this thesis.
  - **Validation:** generated packages can be streamed and played.
- **Operational benefits:**
  - **Package efficiency:** the package is more efficient compared with current legacy Widevine Classic. With proper content ingestion with multiple audios

per video there will be cost savings because of the package is more compacted. Apart from this, adaptive streaming offers more quality by less storage space, customers will be offered with better cinema experiences and the CDN will outperform compared to Widevine Classic.

– **Viability of CMAF in terms of costs:** with the savings generated by Widevine Classic deprecation it is feasible, the cost of keeping the Widevine Classic catalog is higher than the same catalog built with CMAF. With this change the company could reduce the operating expenses and even support new devices.

6.3. Personal Conclusions

Beyond the academic achievements, all the process involving this thesis has been rewarding. This project has given me a real chance to increase my experience working on strategic projects related to media and streaming services. At the beginning I experienced a deep lack on encoding and packaging knowledge, as my background has always been related to software development and networking. After working with the different toolsets, encoders, packagers and in general, any software related to multimedia has given me a deeper knowledge on the field of the company for which I am currently working.

While defining the overall project, it was important to picture the future of streaming, understand which changes were happening and how the company could benefit from them. That was not a common thing I was used to. Then, on the development itself it was not a big issue, as my background helped me on doing the first proofs of concept, the interesting thing was to leverage, understand and differentiate the different options that the market was offering from a strategic point of view. The interest for the solution was not for a short-term, but for a long-term strategy.

Finally, working on a fast-paced company, which schedules projects on a semester basis it is hard to allocate time to study such things like the one carried out in this thesis. Engineers need to understand the company requirements from an economic perspective. That chapter helped me a lot with such tasks, as it is not common when coming from an educational and engineering background.

6.4. Future Work

The results of the thesis point to several interesting directions for future work.

In the case of engineering:

- **Production integration:** Current development has been driven separately from the main encoding tools inside the company. As it has been proven, it is feasible to start working on a production-ready project. This should be scheduled on the company timings and included on the budget estimations.
- **Inclusion of HLS:** With this solution, HLS will finally be introduced in the company. HLS is expected to improve the KPIs of the Apple devices. This will have a direct impact in the customer satisfaction. Current solution for iOS involves transmuxing

on the device which is costly in terms of CPU. The KPIs for this device are lower compared to other ones.

- **Transmuxing in the edge:** Finally, from an engineering point of view, it could be interesting to encrypt and package the medias directly in the CDN edges. If the package is built on the fly, no duplication would be required in the origin, so no doubling of storage cost would happen.

In the case of operations:

- **Improve media ingestion:** CMAF should be economically viable if medias are rich on languages, the cost of adding a new language is less than the cost of the video renditions. Nowadays the content ingestion team sometimes do not ingest all the languages at the same time which ends up on multiple adaptive packages with separated audios, if this is improved the required budget for storage will be reduced, with the cost reduction CMAF could be afforded without a huge investment.
- **Widevine Classic deprecation:** Product team on the company should decide on weather or not deprecate the use of this legacy DRM. It is inefficient from two perspectives, the way it works not supporting multi-resolution streams, and because it is a single technology solution, so medias cannot be shared across different standards, and finally the DRM is on its end-of-life. Deleting all the Widevine Classic medias, it could be feasible to store CMAF medias without bigger expenditures.

6.5. Environmental Impact

Last but not least it is relevant to talk about the environmental impact of the work described in this document. This project analyses a new streaming protocol and studies the viability of its implementation. Although this study has not a direct environmental benefit, its implementation could reduce the amount of storage that is required to handle RakutenTV medias. Having fewer data to be stored reduces the amount of required hard-drives. Typically, hard-drives are prone to fail, if more are required the probability of failure will increase generating waste in terms of hardware.

GLOSSARY

A

AES: Advanced Encryption Standard
AVI: Audio Video Interleave

C

CBC: AES with Cipher Block Chaining
CDM: Content Decryption Module
CDN: Content Delivery Network
CMAF: Common Media Application Framework
CTR: AES with Counter Mode

D

DASH: Dynamic Adaptive Streaming over HTTP
DRM: Digital Rights Management

E

EME: Encrypted Media Extensions

F

fMP4: Fragmented MP4
fps: Frames per second

G

GOP: Group of Pictures

H

HLS: HTTP Live Streaming

I

ISO: International Organization for Standardization
ISOBMFF: ISO Base Media File Format
IV: Initialization Vector

K

KPI: Key Performance Indicator

M

MSS: Microsoft Smooth Streaming

P

PIFF: Protected Interoperable File Format
PoP: Point of Presence

PSSH: Protection System Specific Header

T

TEE: Trusted Execution Environment

V

VoD: Video on Demand

BIBLIOGRAPHY

[1] Motion Picture Association of America “A comprehensive analysis and survey of the theatrical and home entertainment market environment ”. *(THEME) for 2018.* xiii, 1

Report available at https://www.mpaa.org/wp-content/uploads/2019/03/MPAA-THEME-Report-2018.pdf

[2] “Accepted video codecs in MPEG-2 container.” (videolan.org, 2019) 9

Information available at https://wiki.videolan.org/MPEG/#Accepted_video_codecs

[3] “Recommendation for Block Cipher Modes of Operation.”. *National Institute of Standards and Technology*. 11

Information available at https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf

[4] Committee: ISO/IEC JTC 1/SC 29 “Multimedia application format (MPEG-A) - Part 19: Common media application format (CMAF) for segmented media”. *ISO/IEC 23000-19:2018*

Partially available at https://www.iso.org/obp/ui/#iso:std:iso-iec:23000:-19:ed-1:v1:en

[5] Wolenetz, M., Smith, J., Watson, M., Colwell, A., Bateman, A. “Media Source Extensions™”. *W3C Recommendation 17 November 2016*. (w3.org, 2016).

Available at https://www.w3.org/TR/2016/REC-media-source-20161117/

[6] Dorwin, D., Smith, J., Watson. M., Bateman, A. “Encrypted Media Extensions™”. *W3C Recommendation 18 September 2017*. (w3.org, 2017).

Available at https://www.w3.org/TR/2017/REC-encrypted-media-20170918/

[7] Pantos, R., Ed., and W. May “HTTP Live Streaming”. *RFC 8216, DOI 10.17487/RFC8216, August 2017*. 21

Available at https://tools.ietf.org/html/rfc8216

[8] “Secure the delivery of streaming media to devices through the HTTP Live Streaming protocol”. *FairPlay Streaming*.

Available at https://developer.apple.com/streaming/fps/

[9] “About the Common Media Application Format with HTTP Live Streaming”. *HTTP Live Streaming*.

Available at https://developer.apple.com/documentation/http_live_streaming/about_the_common_media_application_format_with_http_live_streaming

[10] Committee: ISO/IEC JTC 1/SC 29 “Part 7: Common encryption in ISO base media file format files”. *ISO/IEC 23001-7:2016*

Partially available at https://www.iso.org/obp/ui/#iso:std:68042:en

[11] Committee: ISO/IEC JTC 1/SC 29 “Dynamic adaptive streaming over HTTP (DASH) — Part 1: Media presentation description and segment formats”. *ISO/IEC 23009-1:2014*.

Available at MPEG-DASHISO/IEC23009-1:2014

[12] John A. Bocharov, Quintin Burns, Florin Folta, Kilroy Hughes, Anil Murching, Larry Olson, Patrik Schnell, John Simmons “Portable encoding of audio-video objects”. *The Protected Interoperable File Format (PIFF)*

Available at https://go.microsoft.com/?linkid=9682897

[13] Rikunov. Andrey “FairPlay Key Security Module (Java)” 32

Available at https://github.com/andreyrikunov/fairplay-ksm

[14] Lin. Eason “FairPlay Key Security Module written in Go (Golang)” 32

Available at https://github.com/easonlin404/ksm

APPENDICES

APPENDIX A. ADAPTIVE STREAMING MANIFESTS

In this appendix there are examples of each streaming manifest explained in *2.State of the art*.

A.1. Microsoft Smooth Streaming (MSS)

Smooth Streaming requires two manifests to work, one will be used by the client to switch between qualities and to request the data corresponding to a time segment. The second, which is being used on the transmuxer will be used to obtain the requested file and return the data corresponding to the queried time segment.

A.1.1. Client manifest

ISMC manifest is used in the client side. It holds all the information related to the stream, it lists all the renditions and time segments. The player will use the bitrate of the rendition and the time segment to request the appropriate byte-data to the transmuxer.

Listing A.1: ISMC Manifest example to be used on the client side

```
<?xml version="1.0" ?>
<SmoothStreamingMedia Duration="80770208365" MajorVersion="2" MinorVersion="0" TimeScale="↩
    10000000">
  <StreamIndex Chunks="4039" Language="eng" Name="audio-eng-mp4a-1" QualityLevels="1" TimeScale=↩
      "10000000" Type="audio" Url="QualityLevels({bitrate})/Fragments(audio-eng-mp4a-1={start ↩
      time})">
    <QualityLevel AudioTag="255" Bitrate="201773" BitsPerSample="16" Channels="2" ↩
        CodecPrivateData="1190" FourCC="AACL" Index="0" PacketSize="4" SamplingRate="48000" />
    <c d="20053302" />
    <c d="20053302" />
    <c d="19839969" />
    <c d="20053302" />
    <c d="20053302" />
    ...
  </StreamIndex>
  <StreamIndex Chunks="4035" DisplayHeight="216" DisplayWidth="384" MaxHeight="1080" MaxWidth="↩
      1920" Name="video" QualityLevels="8" TimeScale="10000000" Type="video" Url="QualityLevels↩
      ({bitrate})/Fragments(video={start time})">
    <QualityLevel Bitrate="355680" CodecPrivateData="00000001674↩
        d400deca0c0efcb808800001f480005dc0078a14cb00000000168e93b3c80" FourCC="H264" Index="0" ↩
        MaxHeight="216" MaxWidth="384" />
    <QualityLevel Bitrate="2079218" CodecPrivateData="00000001674↩
        d401feca07808bf7808800001f480005dc0078c18cb00000000168e93b3c80" FourCC="H264" Index="4" ↩
        MaxHeight="1080" MaxWidth="1920" />
    <c d="20019984" />
    <c d="20019984" />
    <c d="20019984" />
    <c d="20019984" />
    <c d="20019984" />
    ...
  </StreamIndex>
  <Protection>
    <ProtectionHeader SystemID="9a04f079-9840-4286-ab92-e65be0885f95">↩
        XAIAAAEAAQBSAjwAVwBSAE0ASABFAEE...</ProtectionHeader>
  </Protection>
</SmoothStreamingMedia>
```

A.1.2. Server manifest

On the server side, the transmuxer will use the received bitrate to locate the file using the server side manifest. Then, using the requested time segment it will return the corresponding bytes.

Listing A.2: ISMC Manifest example to be used on the server side

```
<?xml version="1.0" ?>
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
  <head>
    <meta content="stream.ismc" name="clientManifestRelativePath"/>
  </head>
  <body>
    <switch>
      <audio src="audio-eng-mp4a-1.isma" systemBitrate="201773">
        <param name="trackID" value="1" valueType="data"/>
        <param name="trackName" value="audio-eng-mp4a-1" valueType="data"/>
      </audio>
      <video src="video-avc1-1.ismv" systemBitrate="355680">
        <param name="trackID" value="1" valueType="data"/>
      </video>
      <video src="video-avc1-2.ismv" systemBitrate="2079218">
        <param name="trackID" value="1" valueType="data"/>
      </video>
    </switch>
  </body>
</smil>
```

A.2. Dynamic Adaptive Streaming over HTTP (DASH)

As it has been explained the DASH protocol does not require a transmuxer, the client is intelligent enough to parse the container and request the appropriate byte-ranges of each segment.

The manifest signals all the files and qualities, and per each file the initialization segment is signaled. The player will obtain this part of the container. With the initialization information the client can compute the byte ranges to fetch or compute which is the segment to obtain when doing trick playing.

Listing A.3: DASH Manifest using the on-demand profile

```
<?xml version='1.0' encoding='utf-8'?>
<MPD xmlns:cenc="urn:mpeg:cenc:2013" xmlns:mspr="urn:microsoft:playready" xmlns="↩
    urn:mpeg:dash:schema:mpd:2011" mediaPresentationDuration="PT2H14M37.021S" minBufferTime="PT2↩
    .00S" profiles="urn:mpeg:dash:profile:isoff-on-demand:2011" type="static">
    <BaseURL>https://prod-origin-pmd.cdn.server.com/example-media/</BaseURL>
    <Period>
        <AdaptationSet maxHeight="1080" maxWidth="1920" mimeType="video/mp4" minHeight="216" ↩
            minWidth="384" par="16:9" sar="1:1" segmentAlignment="true" startWithSAP="1">
            <ContentProtection cenc:default_KID="534042b7-4404-9f3f-a95b-9301bf4b079e" ↩
                schemeIdUri="urn:mpeg:dash:mp4protection:2011" value="cenc" />
            <ContentProtection schemeIdUri="urn:uuid:9a04f079-9840-4286-ab92-e65be0885f95">
                <mspr:pro>XAIAAAEAAQBSAjwAVwBSAE0ASABFAEE...</mspr:pro>
            </ContentProtection>
```

```
            <ContentProtection schemeIdUri="urn:uuid:edef8ba9-79d6-4ace-a3c8-27dcd51d21ed">
                <cenc:pssh>AAAAanBzc2gAAAAA7e+...</cenc:pssh>
            </ContentProtection>
            <Representation bandwidth="355680" codecs="avc1.4D400D" frameRate="933747/38945" ↩
                 height="216" id="video-avc1-1" scanType="progressive" width="384">
                <BaseURL>video-avc1-1.ismv</BaseURL>
                <SegmentBase indexRange="1410-49861">
                    <Initialization range="0-1409" />
                </SegmentBase>
            </Representation>
            <Representation bandwidth="2079218" codecs="avc1.4D401F" frameRate="933747/38945" ↩
                 height="1080" id="video-avc1-2" scanType="progressive" width="1920">
                <BaseURL>video-avc1-2.ismv</BaseURL>
                <SegmentBase indexRange="1410-49861">
                    <Initialization range="0-1409" />
                </SegmentBase>
            </Representation>
        </AdaptationSet>
        <AdaptationSet lang="eng" mimeType="audio/mp4" segmentAlignment="true" startWithSAP="1">
            <ContentProtection cenc:default_KID="534042b7-4404-9f3f-a95b-9301bf4b079e" ↩
                 schemeIdUri="urn:mpeg:dash:mp4protection:2011" value="cenc" />
            <ContentProtection schemeIdUri="urn:uuid:9a04f079-9840-4286-ab92-e65be0885f95">
                <mspr:pro>XAIAAAEAAQBSAjwAVwBSAE0ASABFAEE...</mspr:pro>
            </ContentProtection>
            <ContentProtection schemeIdUri="urn:uuid:edef8ba9-79d6-4ace-a3c8-27dcd51d21ed">
                <cenc:pssh>AAAAanBzc2gAAAAA7e+...</cenc:pssh>
            </ContentProtection>
            <Representation audioSamplingRate="48000" bandwidth="201773" codecs="mp4a.40.2" id="↩
                 audio-eng-mp4a-1">
                <AudioChannelConfiguration schemeIdUri="↩
                     urn:mpeg:dash:23003:3:audio_channel_configuration:2011" value="2" />
                <BaseURL>audio-eng-mp4a-1.isma</BaseURL>
                <SegmentBase indexRange="1344-49843">
                    <Initialization range="0-1343" />
                </SegmentBase>
            </Representation>
        </AdaptationSet>
    </Period>
</MPD>
```

A.3. HTTP Live Streaming (HLS)

On HLS there are multiple manifests holding different kind of information but all of them are processed locally on the client side.

The master manifest holds the information related to all the renditions of a package, each rendition points to a secondary manifest that signals all the byte ranges contained in the video file associated with that rendition.

For the master manifest refer to listing A.4 and A.5 for the stream manifest.

Listing A.4: HLS master manifest that holds all the renditions of a package

```
#EXTM3U

#EXT-X-MEDIA:TYPE=AUDIO,GROUP-ID="stereo",LANGUAGE="en",NAME="English",DEFAULT=YES,AUTOSELECT=↩
    YES,URI="audio-eng-mp4a-1.m3u8"

#EXT-X-STREAM-INF:BANDWIDTH=355680,AVERAGE-BANDWIDTH=355680,CODECS="avc1.4D400D",RESOLUTION=384↩
    x216
video-avc1-1.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=2079218,AVERAGE-BANDWIDTH=2079218,CODECS="avc1.4D401F",RESOLUTION↩
    =1920x1080
video-avc1-2.m3u8
```

Listing A.5: HLS stream manifest that holds all the byte ranges of a single video

```
#EXTM3U
#EXT-X-VERSION:6

#EXT-X-TARGETDURATION:6
#EXT-X-PLAYLIST-TYPE:VOD
#EXT-X-MAP:URI="video-avc1-2.mp4",BYTERANGE="1409@0"
#EXT-X-KEY:METHOD=SAMPLE-AES,URI="key2.bin",IV=0xDD439B085C1357B51889720DB8D25083,KEYFORMAT="↩
    identity"
#EXTINF:6.000,
#EXT-X-BYTERANGE:1680937@49861
video-avc1-2.mp4
#EXTINF:6.000,
#EXT-X-BYTERANGE:1659882
video-avc1-2.mp4
#EXTINF:6.000,
#EXT-X-BYTERANGE:1467634
video-avc1-2.mp4
#EXTINF:6.000,
#EXT-X-BYTERANGE:1456582
video-avc1-2.mp4
#EXTINF:6.000,
#EXT-X-BYTERANGE:1688090
video-avc1-2.mp4
```

APPENDIX B. CMAF-TOOLS DOCKER IMAGE

Docker has been used to simplify all the encoding and packaging process. Having an image containing all the necessary tools makes the software portable between computers, and avoids installing dependencies and leaving a dirty environment. Every time a docker image is executed (through a container) it starts with a clean and reproducible environment. It simplifies all the environment preparation for this thesis.

B.1. Installed software

The docker image that has been built contains all the required software to encode and package the media files. The following is the software that can be found on it:

- **Encoding:** ffmpeg and mediainfo have been installed directly from Debian 9 (Stretch) repositories. The ffmpeg version of Debian stable is fair enough for the tests carried out in this thesis. For production, probably it will be necessary to build it from the source, applying the patches that the company is using.
- **Packaging:** both Bento4 and Shaka-Packager have been installed, Shaka-Packager is compiled from its source whereas Bento4 can be added from the attached binaries located in their site.

The usage of docker is out of the scope of this thesis, but there are several guides that helps on start working with this tool. Refer to the following site for further information `https://docs.docker.com/get-started/`.

B.2. Repositories

The Dockerfile has been uploaded into a public repository, where it can be fetched and build on a computer o server. It can be reached in the following GitHub repository: `https://github.com/GerardSoleCa/cmaf-tools`. On the other hand, the prebuilt image can be fetched from the docker public repository in `https://hub.docker.com/r/gerardsoleca/cmaf-tools`.

B.2.1. Dockerfile

The following listing shows the content of the Dockerfile, how the software is included and built, so the image contains all the appropriate software.

Listing B.1: Dockerfile to build the gerardsoleca/cmaf-tools

```
FROM debian:9 as builder

RUN apt-get update && apt-get install -y python build-essential curl git

# Shaka Packager
```

```
# Install depot_tools.
RUN git clone https://chromium.googlesource.com/chromium/tools/depot_tools.git
ENV PATH $PATH:/depot_tools

WORKDIR shaka_packager
RUN gclient config https://www.github.com/google/shaka-packager.git --name=src --unmanaged
RUN gclient sync
RUN cd src && ninja -C out/Release

# Generate lightweight image
from debian:9 as final

ENV PATH "$PATH:/opt/bento/bin/:/opt/shaka_packager/bin/"
RUN mkdir -p /opt/shaka_packager/bin

RUN apt-get update && apt-get install -y unzip ffmpeg vim python mediainfo

# Copy compiled binaries
COPY --from=builder /shaka_packager/src/out/Release/packager \
                    /shaka_packager/src/out/Release/mpd_generator \
                    /shaka_packager/src/out/Release/pssh-box.py \
                    /opt/shaka_packager/bin/
COPY --from=builder /shaka_packager/src/out/Release/pyproto /usr/bin/pyproto

# Bento4
ARG BENTO4_VERSION=1-5-1-628
ADD http://zebulon.bok.net/Bento4/binaries/Bento4-SDK-$BENTO4_VERSION.x86_64-unknown-linux.zip /↩
    tmp/bento.zip
RUN unzip /tmp/bento.zip -d /opt && mv /opt/Bento4-SDK-$BENTO4_VERSION.x86_64-unknown-linux /opt↩
    /bento
```