UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Implementation of a Secure DASH Module

A Degree Thesis

Submitted to the Faculty of the

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

by

Garcia Tello, Daniel

In partial fulfilment

of the requirements for the degree in
TELECOMMUNICATIONS SCIENCE AND TECHNOLOGY ENGINEERING

Advisor: Delgado Merce, Jaime M.

Barcelona, June 2016

Abstract

This project is to implement a security module within the MPEG-dash standard, an adaptive bitrate streaming that allows the streaming of multimedia content on the Internet delivered from conventional HTTP web servers.

It has worked based on a client open-source already deployed, from this has created a client-server HTTP architecture, to which have been uploaded test samples of multiple content types that complies with the standard.

All the content and functionality of the client has been tested, it has made an analysis of the code and its implementation without any modification, and it has proceeded to perform a security module design taking into account the current state of the community and the various proposals which have emerged.

Then, a module has been designed which allows to detect when it is necessary to process the content, in which files, obtain the necessary information to get and decrypt the multimedia content, and delivery to the player, keeping the rest of functionalities of the protocol/client.

Finally, it has been tested both, the accomplishment of the protocol and the security toward the content.

Resum

Aquest projecte té com a finalitat implementar un mòdul de seguretat dins de l'estàndard MPEG-DASH, una tècnica de streaming adaptatiu que manté la qualitat del contingut multimèdia proporcionat per un servidor HTTP gràcies al canvi de taxa de bits.

S'ha treballat partint d'un client de codi obert ja implementat. A partir d'aquest s'ha configurat una arquitectura client servidor HTTP. En el servidor s'ha pujat una mostra de diversos tipus de contingut que compleixen l'estàndard per poder testejar.

S'ha testejat tot el contingut i les funcionalitats del client, s'ha fet una anàlisi del codi i la seva execució sense introduir cap modificació, i s'ha procedit a realitzar un disseny del mòdul de seguretat tenint en compte l'estat actual de la comunitat i les diferents propostes que han anat sorgint.

S'ha dissenyat un mòdul el qual permet detectar quan és necessari processar el contingut, en què arxius, obtenir la informació necessària per adquirir i desencriptar el contingut multimèdia i lliurar-los al reproductor, mantenint la resta de funcionalitats del protocol.

Finalment s'ha testejat tant el compliment del protocol com la seguretat cap al contingut.

Resumen

Este proyecto tiene como finalidad implementar un módulo de seguridad dentro del estándar MPEG-DASH, una técnica de streaming adaptativo que mantiene la calidad del contenido multimedia proporcionado por un servidor HTTP gracias al cambio de tasa de bits.

Se ha trabajado partiendo de un cliente de código abierto ya implementado. A partir de este se ha configurado una arquitectura cliente servidor HTTP. En el servidor se ha subido una muestra de varios tipos de contenido que cumplen el estándar para poder testear.

Se ha testeado todo el contenido y las funcionalidades del cliente, se ha hecho un análisis del código y su ejecución sin introducir ninguna modificación, y se ha procedido a realizar un diseño del módulo de seguridad teniendo en cuenta el estado actual de la comunidad y las diferentes propuestas que han ido surgiendo.

Se ha diseñado un módulo el cual permite detectar cuando es necesario procesar el contenido, en que archivos, obtener la información necesaria para adquirir y desencriptar el contenido multimedia y entregarlos al reproductor, manteniendo el resto de funcionalidades del protocolo.

Finalmente se ha testeado tanto el cumplimiento del protocolo como la seguridad hacia el contenido.

Acknowledgements

I want to give my thanks to my advisor Jaime Delgado, for giving me the opportunity of doing my bachelor thesis about its development being involved in MIPAMS project on the DMAG research group and for all advices provided to improve my own project. Moreover, I also want to give my thanks to Silvia Llorente for their support in all meetings.

Revision history and approval record

Revision	Date	Purpose
0	11/07/2016	Document creation
1	19/09/2016	Document revision
2	26/09/2016	Document revision
3	27/09/2016	Document revision

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Garcia Tello, Daniel	daniel.garcia.tello@alu-etsetb.upc.edu
Delgado Merce, Jaime M.	jaime.delgadoa@ac.upc.edu
Llorente Viejo, Silvia	silviall@ac.upc.edu

Written by:		Reviewed and approved by:	
Date	11/07/2016	Date	27/09/2016
Name	Garcia Tello, Daniel	Name	Delgado Merce, Jaime M.
Position	Project Author	Position	Project Supervisor

Table of contents

List of Figures

List of Tables:

1. Introduction

1.1. Statement of purpose

The project is carried out at UPC DMAG (Distributed Multimedia Applications Group) research group of the Computer Architecture Department at the Universitat Politècnica de Catalunya (UPC). The current research and development activities mainly deal with the creation, management and distribution of multimedia content in a secure and interoperable way. Research topics include Digital management of rights, Metadata interoperability, Multimedia search, Security and privacy and Ontologies and semantics. The main application areas are Content life cycle, E-Health, Online social networks and E-Services. A key issue in all the previous topics and areas is international standardization.

This project consists on adapting, improving and integrating open source code. This module will provide basic functionality of rights management and security aspects within the compliance with the DASH standard. A demonstration application should be developed.

The project main goals are:

1. Setup a working client-server configuration to operate with a DASH protocol.
2. Implement a module integrating a DASH implementation in a secure system for the management and interchange of multimedia content.
3. Develop a demonstration application.

1.2. Requirements and specifications

Project requirements:

- Correct transmission of multimedia using DASH protocol.
- Deny access to the non-permitted users.
- Allow access to the specific permitted users
- Compliance with the standard

1.3. Methods and procedures

This project consists in adding functionality to a software already implemented in compliance with the standard DASH.

First, we will look at the standard and the solutions that have implemented, which we are going to provide the base with the main features of this Protocol, from which to begin to work and to add our module. Thus, seek the solution that will allow us to work with the standard, make modifications easily, and to allow us to perform a tested on him.

Thus, once obtained the software base is will proceed to analyse the state of the Protocol and the community by focusing on the security implementations.

Then, it will proceed to implement the security module in our software and finally tested both the compliance with standard as the functionality of the module and the correct operation in regard to security.

1.4. Work plan

The following points describes the work plan, with the tasks involved in the project, its duration and the detailed procedure.

1.4.1. Tasks

Project: Setup DASH Server/Client	WP ref: 1	
Major constituent: SW	Sheet 1 of 4	
Short description: Setup a working server and client configuration to operate with a DASH protocol, adapting, improving and integrating open source code.	Planned start date: 03/03/16 Planned end date: 08/05/16	
	Start event: 03/03/16 End event: 03/03/16	
Internal task T1: Server implementation Internal task T2: Client implementation Internal task T3: Test system	Deliverables:	Dates:

Project: Implementation of a Secure DASH Module	WP ref: 2	
Major constituent: SW	Sheet 2 of 4	
Short description: Implement a module to provide basic DASH (client and server) functionality focused to given the rights management and security aspects.	Planned start date: 26/05/16 Planned end date: 28/08/16	
	Start event: 26/05/16 End event: 28/08/16	
Internal task T1: Creation of a Module Internal task T2: Implementation of Security Information Internal task T3: Implementation of a working Security Module	Deliverables:	Dates:

Project: Demo Application	WP ref: 3	
Major constituent: Simulation	Sheet 3 of 4	
Short description: Develop a demonstration application.	Planned start date: 28/08/16 Planned end date: 27/09/16	
	Start event: 28/08/16 End event: 27/09/16	
	Deliverables:	Dates:
Project: Thesis	WP ref: 4	
Major constituent: Redaction	Sheet 4 of 4	
Short description: Drafting of the document TFG Thesis final report.	Planned start date: 15/09/16 Planned end date: 28/09/16	
	Start event: 15/09/16 End event: 28/09/16	
	Deliverables: Final Report	Dates: 28/09/16

Table 1 Work Plan Tasks

1.4.2. Milestones

WP#	Task#	Short title	Milestone / deliverable	Date (week)
1	1	Server	Milestone	30/03/2016
1	2	Client	Milestone	25/04/2016
1	3	Test	Milestone	03/05/2016
1,2	-	Critical Review	Deliverable	28/09/2016
2	1	Module	Milestone	15/06/2016
2	2	Security Info	Milestone	10/07/2016
2	3	Working Module	Milestone	11/08/2016
3	-	Application	Milestone	15/09/2016
4	-	Thesis	Deliverable	28/09/2016

Table 2 Work Plan Milestones

1.4.3. Time Plan

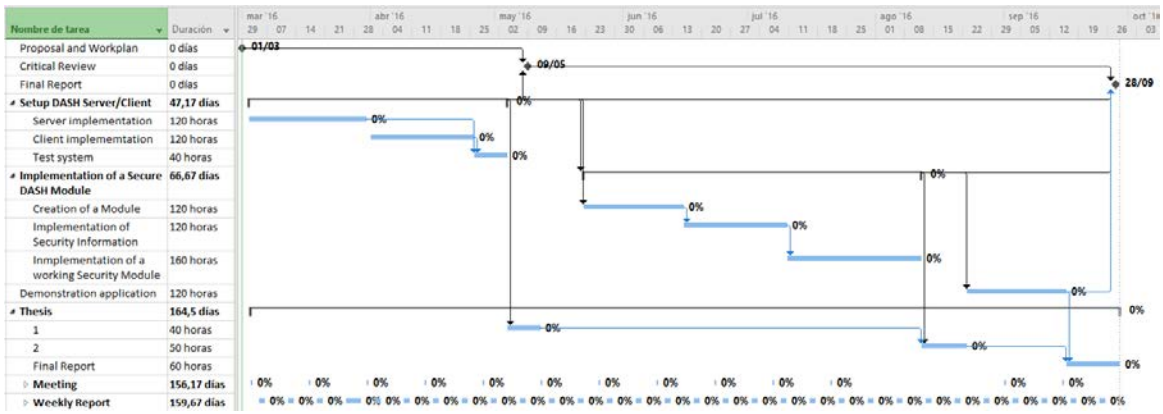

Figure 1 Work Time Plan

1.5. Issues

Throughout the implementation of the project, it has arisen mainly two disadvantages. In the first place, the initial client chosen has proved to be a bad choice, it depended on external libraries and these are not controlled, which it has given rise to inconsistencies that were not predicted and at the end it was finished discarding the client due to the constant modifications to be made and which we do not guarantee a later stable behaviour.

Secondly, we found as counterparties in the second client that, despite that worked correctly, when implementing the encrypted on the server side and the deciphered in the part of the client, encryption and decryption are not symmetrical, due to the very different origin of the information. Thus, has had to find several solutions to achieve a correct deciphered and symmetry, highlighting as the main problem from libraries totally unrelated to be implemented in different languages (server and client).

2. State of the art of the technology used or applied in this thesis:

Dynamic Adaptive Streaming over HTTP, or DASH, is a technique of stream of multimedia content through the internet using conventional HTTP web servers.

DASH has as its purpose the reproduction of multimedia content to maintain an optimal bitrate. This is accomplished by segmenting the content in files, so that they obtain segments in several resolutions, with which we have available different bitrates. This information of the segments is stored in a media presentation description or MPD so that the client can choose which segment sue in order to maintain an optimal bitrate, i.e., an optimal quality, in each moment.

Therefore, a client dash to play multimedia content first it needs to obtain the MPD with the information of the segments and, secondly, calculate the bandwidth to request the appropriate segment and maintain the optimal bandwidth.

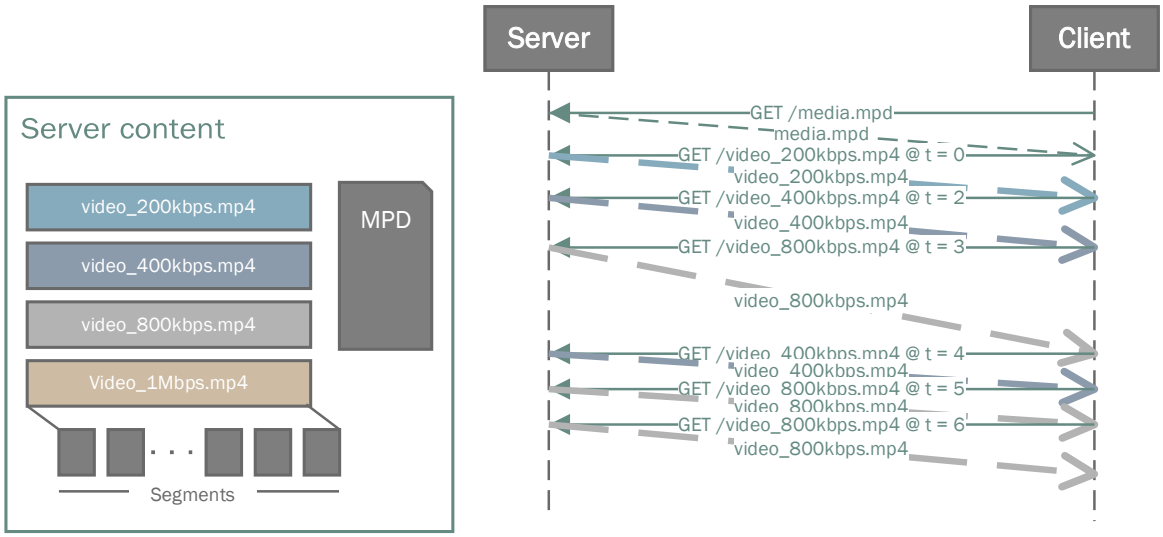

Figure 2 DASH Data Exchange

2.1. DASH Client

DASH client can be understood as a HTTP client that implement the features required to process the Media Presentation (MPD document and Segments).

Mainly, the client is responsible for obtaining the MPD using the HTTP protocol, take out the information and process it for your use, and then, go getting the proper segments to maintain an optimal bitrate and reproduce and display the multimedia content.

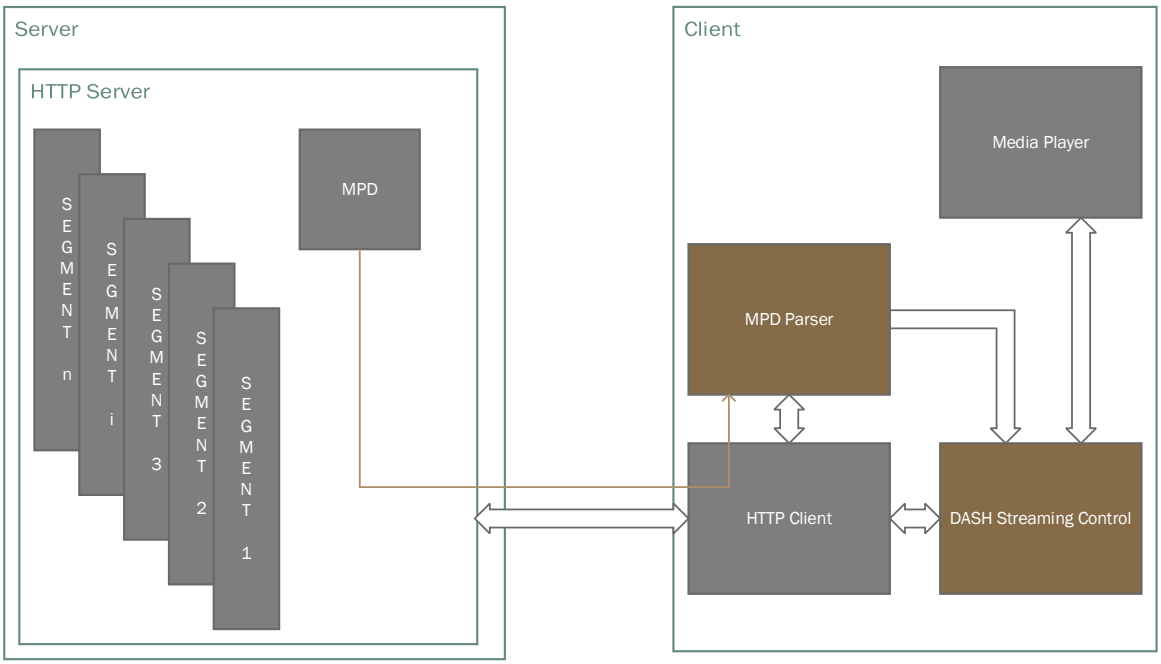

Figure 3 DASH Main Structure

Basically, a DASH client must be able to manage the following aspects:

- Selection of the appropriate Adaptation Sets based on descriptors and other attributes.
- Initial selection of one Representation within each adaptation set.
- Download of (Sub)Segments at the appropriate time.
- Synchronization of different media components from different Adaptation Set.
- Seamless switching of representations within one Adaptation Set.
- Support of HTTP GET and partial GET requests to download Segments and Subsegments.
- Three different addressing schemes: number and time-based templating as well as byte range based requests.
- Support of metadata as provided in the MPD and Segment Index.
- Download of Media Segments, Initialization Segments and Segment Index.
- ISO BMFF parsing.
- Synchronized presentation of media components from different Adaptation Sets.
- Switching of video streams at closed GOP boundaries.

2.2. Media Presentation Description

The MPEG-DASH Media Presentation Description (MPD) is an XML document containing information about media segments, their relationships and information necessary to choose between them, and other metadata that may be needed by clients.

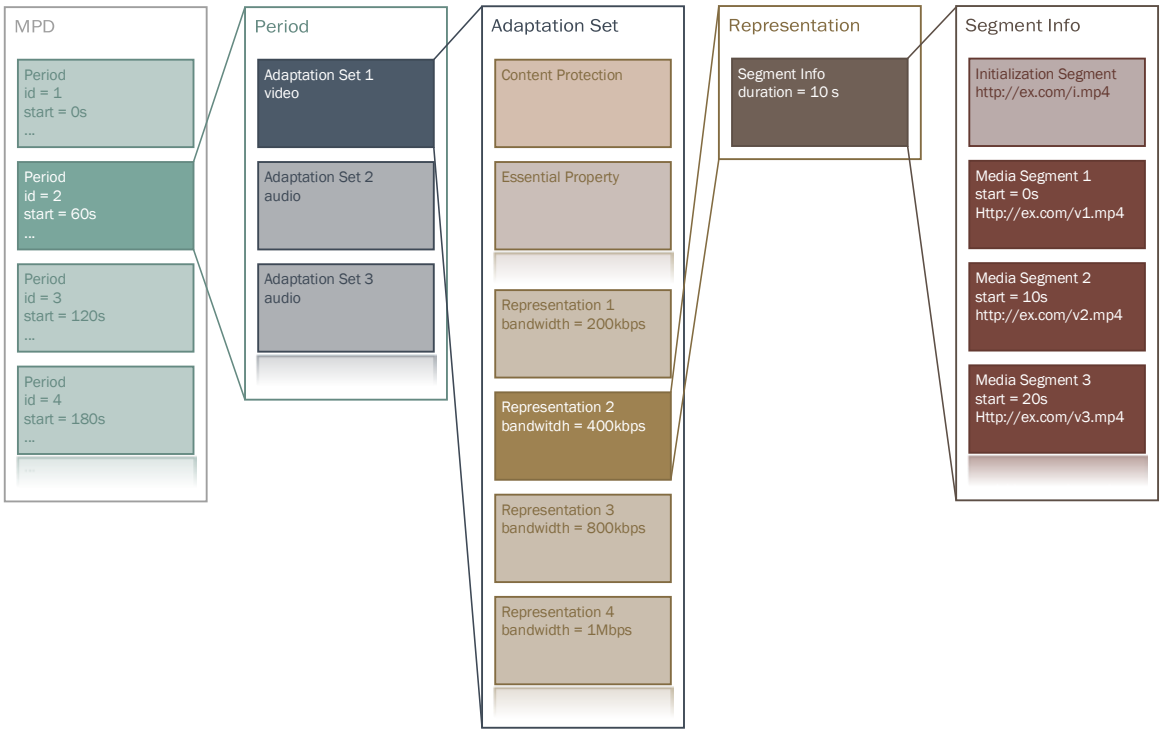

Figure 4 Basic Structure of the Data

Looking in more detail we can focus on two fields that are going to be useful, in the first place this defined element *ContentProtection* which provides the mechanisms to signal the use of DRM, already well defined.

In the second place we have the element *EssentialProperty*, which is defined as follows: *"EssentialProperty specifies information about the containing element that is considered essential by the Media Presentation author for processing the containing element."*, which provides us with a mechanism to specify the information that we needed.

2.3. Content Protection and Security

In regard to the implementation of security, *dashif* gives us two implementation guidelines, one addressed to the DRM protocol and encryption, which is well defined, and another for access control using tokens.

2.3.1. Token-based Access Control

The Access Token shall be transported in a HTTP header field when communicated in a HTTP 2xx Successful message, or in a query string parameter when communicated in a HTTP 3xx Redirection message or in a HTTP request.

The field access-token is a string containing the base64-encoded Access Token. The query string parameter name shall be "*dash-if-ietf-token*" whose value is the base64-encoded Access Token.

To be able to exchange this information are defined three different protocols, the first two mechanisms, the client is independent of what is the nature of the testimony of access and its function, which simply it is treated as a string that you need to pass along with the necessary HTTP requests. The last, the client has to manage a separate protocol.

- HTTP-based instantiation:

This mechanism instantiates the initial Access Token from regular DASH operations. The Access Token may be delivered in a HTTP header of a MPD response, a segment response, etc... In practice, the HTTP server delivers the requested resource along with the Access Token. In addition, the MPD author signals in the MPD how to extract the Access Token and from which HTTP responses. To this end, the *ExtUrlQueryInfo* shall be used and should be configured according to the mechanism expected by the MPD author.

The example MPD below signals the presence of the Access Token in the HTTP responses for MPD requests (*@headerParamSource* attribute) inside the HTTP custom header called “DASH-IF-IETF-Token” and instructs the DASH Client to insert this Access Token into segment and MPD requests within the “dash-if-ietf-token” query string parameter.

```
1 <EssentialProperty schemeIdUri="urn:mpeg:dash:urlparam:2016:querystring"
  xmlns:up="urn:mpeg:dash:schema:urlparam:2016">
2    <up:ExtUrlQueryInfo headerParamSource="mpd" includeInRequests="segment mpd"
     queryTemplate="dash-if-ietf-token=$header:DASH-IF-IETF-Token$"/>
3 </EssentialProperty>
```

Table 3 Extract of a MPD with HTTP-based Instantiation

- MPD-based Access Token instantiation

The second mechanism instantiates the initial Access Token from an appropriate XML element in the MPD. It is encouraged to secure the delivery of the MPD in that case via a secure transport protocol to prevent illegal clients from intercepting the Access Token.

The *@queryString* attribute of either the *UrlQueryInfo* or *ExtUrlQueryInfo* descriptors shall be used to carry the Access Tokens.

```
1 <EssentialProperty schemeIdUri="urn:mpeg:dash:urlparam:2016:querystring"
  xmlns:up="urn:mpeg:dash:schema:urlparam:2016">
2    <up:ExtUrlQueryInfo includeInRequests="mpd segment"
     queryString="token=nitfHRCrtziwO2HwPfWw~yYD" queryTemplate="dash-if-ietf-
     token=$query:token$"/>
3 </EssentialProperty>
```

Table 4 Extract of a MPD with MPD-based Access Token Instantiation

- External Access Token instantiation

The last mechanism instantiates the Access Token through an external protocol. In this case, the MPD signals the protocol to be used by the application to retrieve the initial Access Token. When the application recognizes one of the signalled protocols, it executes the corresponding protocol as specified by the scheme. Since out-of-scope of this specification, it is expected that the application implements the different steps of the protocol that leads to the retrieval of token.

2.3.2. Encryption and DRM Signaling.

Content Keys and DRM Signaling, also known as content protection information, need to be created and exchanged between some system entities when preparing content. The flow of information are of very different nature depending on where Content Keys are created and also depending on the type of content that can be either On-Demand or Live.

This implementation of security using DRM is well-defined and provides us with a robust configuration and at the same time extremely complex.

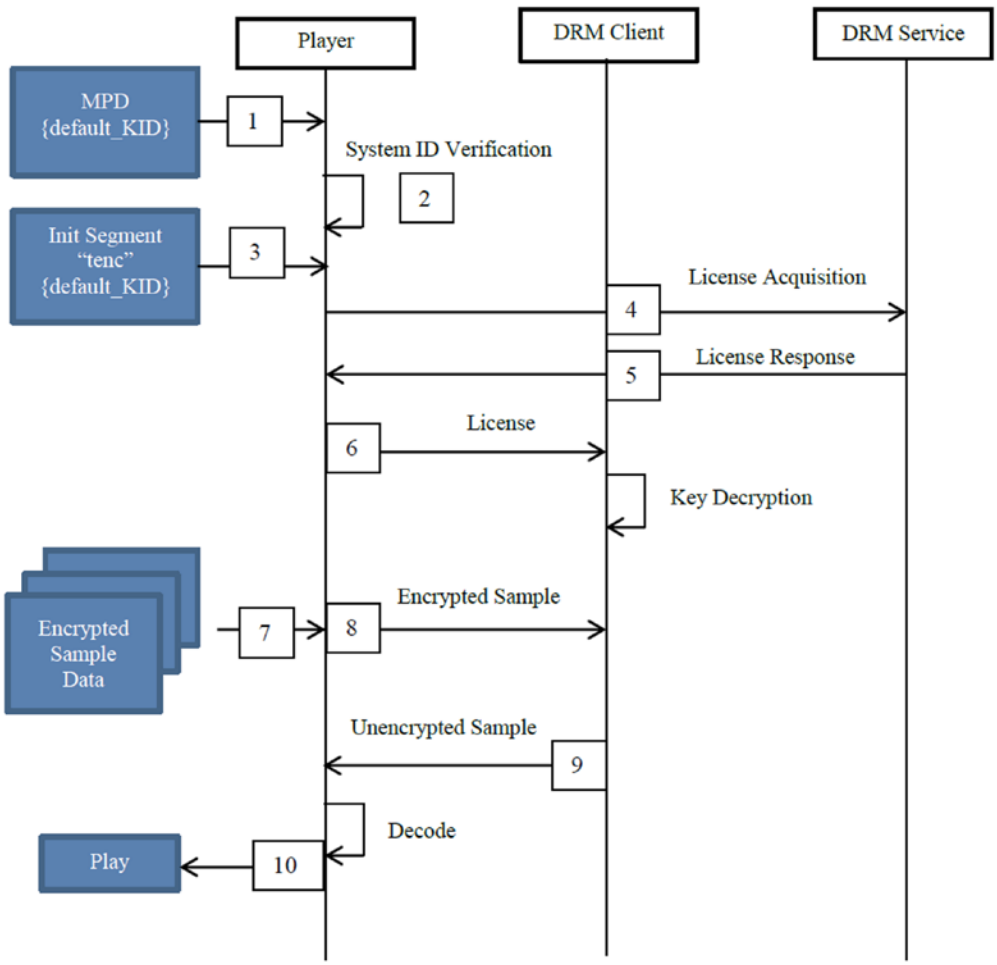

Figure 5 Basic Data Exchange on a DRM configuration

3. Methodology / project development:

In order to develop a system for the protection of content delivered through the DASH protocol, and that therefore conforms to the specifications defined in this protocol, it has followed the next methodology.

In the first place, it has done an approximation to the standard dash with a commissioning of a client already implemented, with such testing the functionality from a solid base with the smallest possible errors. Then, we proceed to test and analyse the client, server, and samples of the content provided by the same client. In third place has obtained a software to generate MPD and the segmentation and thus be able to generate new own content, edit and validate the MPD resulting to make sure that you comply with the standard. Below has deployed a server to be able to work with the modified content. Finally, it has proceeded to implement the security module and test it.

3.1. Set up DASH client

First, has been sought, between all the alternatives available, a software open source, and modular, which allows us to make modifications without taking into account a large portion of code, and will allow us to test our changes.

3.1.1. C++ client

Among all the alternatives to the scope, has been chosen in the first instance a solution implemented in C++ language, which as the main feature allows us to the possibility of compiling and obtain an independent client, compared to the other options that are implemented in JavaScript, an interpreted language that depends on a browser. As well as it is an open-source library and an object orient interface to the MPEG-DASH standard, which gives us the addition of a module.

In regard to the architecture of this client, it handles the download and XML parsing of the MPD, which provides an interface for application developers based on that information. It encapsulates the XML parsing process of the MPD and thereof it provides an object oriented interface to the MPD and on the other hand it handles the download process of the media segments. The library is cross platform buildable including Windows, Linux and Mac. It implements the full *ISO/IEC MPEG-DASH* standard according to *ISO/IEC 23009-1*, and.

Internally the library uses *libcurl* for the download of the individual segments and *libxml* to implements the XML parser.

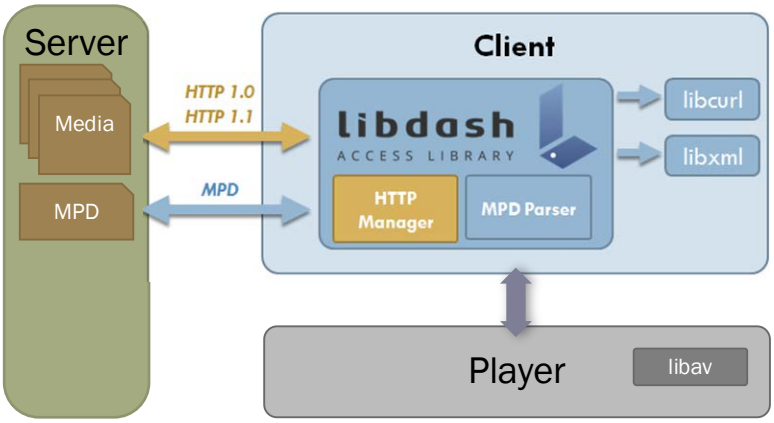

Figure 6 Libdash Block Diagram

In addition, the same client has an independent player to test it, the *QTSamplePlayer*, which depends on the *QT* software to build the GUI and the *libav* library for the audio and video processing tools, both open source.

However, at the time of compiling it have been emerging problems, given that this client to be compiled also has its disadvantages, since it depends on the architecture on which we are. In this case which it has prevented us from compile successfully and therefore use that client has been the third-party software, since for this client have been used external libraries, which have been changing and does not fit perfectly to the client.

```
:/libdash/libdash/build/bin$ ./libdash_networkpart_
test
****************************************
* Download files with external HTTP 1.0 *
****************************************
Testing download of text file:  finished!
Testing download of video file: finished!

****************************************
* Download files with external HTTP 1.1 *
****************************************
Testing download of text file:  finished!
Testing download of video file: finished!
```

Figure 7 Libdash Test Results

Once compiled successfully the client, with the corresponding libraries of which it depends (*libxml*, *libcurl*, *zlib*), we have proceeded to test. To do this, we have a player already available, which it depends on the library *libav*. At the time of compiling it, it have been appearing in compile errors related to this library *libav*, which it has changed since that was designed by the client.

Once managed to compile without errors and start the player, it has been appearing errors both when requesting the MPD and processed as to play the content, which occasionally even it make break executions.

```
/libdash/libdash/qtsampleplayer/build$ ./qtsampleplayer
http://tvnlive.dashdemo.edgesuite.net/live/manifest.mpd:4: parser error : Opening and ending tag mismatch: p line 2 and BODY
</BODY></HTML>
          ^
[mov,mp4,m4a,3gp,3g2,mj2 @ 0x14704a0] Could not find codec parameters for stream 0 (Video: none (avc3 / 0x33637661), none, 1280x720, 44 kb/s): unknown codec
Consider increasing the value for the 'analyzeduration' and 'probesize' options
[aac @ 0x146a8a0] element type mismatch 1 != 0
[aac @ 0x146a8a0] element type mismatch 1 != 0
[aac @ 0x146a8a0] element type mismatch 1 != 0
[aac @ 0x146a8a0] element type mismatch 1 != 0
[aac @ 0x7efe60029e20] element type mismatch 1 != 0
```

Figure 8 Sample of some of the Errors Generated by the Player

After many modifications to try to adjust the correct version of each module, and try to rebuild the client with the libraries used in the date on which was successfully implemented, it was decided to abandon this solution due to the constant execution errors and implementation, since the purpose of adopting a client already implemented was to obtain a base with the smallest possible errors.

3.1.2. JavaScript client

Thus as a second option, it has been adopted a client implemented in JavaScript, leveraging the Media Source Extensions API set as defined by the W3C. We will us a language not compiled, which although it depends on a browser in this case, however this client in particular is both codec and browser agnostic. That is what are going to allows us to run the code without problems since it does not depend on external libraries, using only node.js, which lets us to run part of the code on the server side. In addition, we will have the advantage of working with an open source JavaScript library for the playback of DASH.

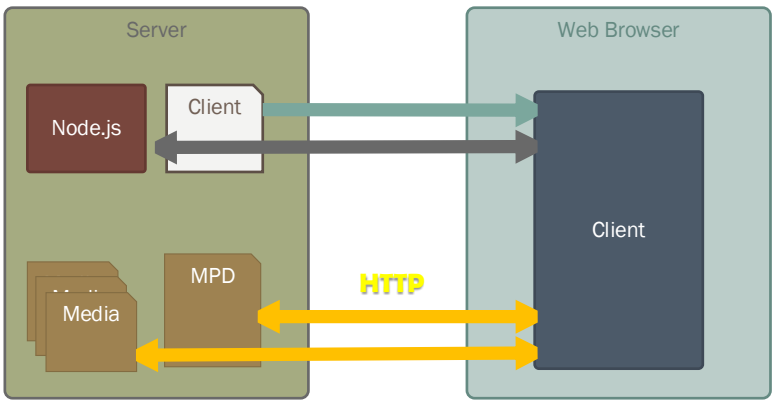

Figure 9 DASH.JS Block Diagram

Finally, this client allows us to choose a version to run from their own server already running or the possibility to run it in our own machine, i.e. the possibility of introducing modifications.

3.2. Analysing DASH System

Firstly, it has proceeded to play a test content already given, in the client provided from the server itself, in order to analyse the execution of each module. We have focused on the MPD analysis or parse, and the obtaining of the media files.

Due to the variety of content that can be transmitted using the DASH, we can create different structures and therefore the analysis of the parsing is done too broad. It is for this reason that we have focused on the analysis of the processing of a MPD to play content on demand, with only one Period, and an Adaptation Set for audio and one for video.

To play such content, the client processes the MPD and request to the HTTP server, using GET method, specific sets of bytes depending on the time of reproduction, the quality of the signal, the user interaction with the playback controls and of course the information extracted from the MPD.

3.2.1. Set up media server

Due to this client allows us to choose an online version and use any server where is located the MPD, the next step is to set up an HTTP server, generate our own content and MPD and check the client example.

To do this, in the first place it has configured an Apache server on which it will allocate the MPD and their respective content.

Apache has been chosen to be basically the web server software most used in the world, which we provide a wide community to find support to future problems that we can find, and which provides us some common language interfaces support (Perl, Python, Tcl, and PHP) and have some popular authentication modules, which all this provides us with widely we need to deploy our server and use possible tools that we need for our purpose.

In second place, the content has been generated through an application that splits a video given and generates the MPD corresponding to this content. We have several already deployed applications that perform this task, transcoding correctly our multimedia content, segment and generating a MPD that complies with the standard.

We have chosen a segmenter and generator of MPD on-line since that allows us to generate reliable content without giving rise to errors produced by us, given that nor do we need, in a principle, focus on the multimedia content or the way to generate this, if not in the MPD that it is where we will focus in order to implement the module according to the standard.

```
mpd ..
  @xmlns:ns2: http://www.w3.org/1999/xlink
  @xmlns: urn:mpeg:dash:schema:mpd:2011
  @xmlns:bitmovin: http://www.bitmovin.net/mpd/2015
  @id: 67bc29d8-59f3-49e6-b8ae-c63bd8ccbfe0
  @profiles: urn:mpeg:dash:profile:isoff-main:2011
  @type: static
  @availabilityStartTime: 2016-04-11T16:24:19.000Z
  @publishTime: 2016-04-11T16:24:40.000Z
  @mediaPresentationDuration: P0Y0M0DT0H14M48.000S
  @minBufferTime: P0Y0M0DT0H0M1.000S
  @bitmovin:version: 1.7.0
  period ..
    adaptationset ..
      @mimeType: video/mp4
      segmenttemplate
        @media: ../video/$RepresentationID$/segment_$Number$.m4s
        @initialization: ../video/$RepresentationID$/init.mp4
        @duration: 96000
        @startNumber: 0
        @timescale: 24000
      representation
        @id: 816_4800000
        @bandwidth: 4800000
        @width: 1920
        @height: 816
        @frameRate: 24
        @codecs: avc1.640028
      representation
        @id: 544_2400000
        @bandwidth: 2400000
        @width: 1280
        @height: 544
        @frameRate: 24
        @codecs: avc1.64001F
      representation
        @id: 362_1200000
        @bandwidth: 1200000
        @width: 854
        @height: 362
        @frameRate: 24
        @codecs: avc1.64001E
      representation
        @id: 272_800000
        @bandwidth: 800000
        @width: 640
        @height: 272
        @frameRate: 24
        @codecs: avc1.640015
      representation
        @id: 180_400000
        @bandwidth: 400000
        @width: 426
        @height: 180
        @frameRate: 24
        @codecs: avc1.64000D
    adaptationset ..
      @lang: en
      @mimeType: audio/mp4da
      @codecs: mp4a.40.2
      @bitmovin:label: english stereo
      audiochannelconfiguration
        @schemeIdUri: urn:mpeg:dash:23003:3:audio_channel_configuration:2011
        @value: 2
      segmenttemplate
        @media: ../audio/$RepresentationID$/segment_$Number$.m4s
        @initialization: ../audio/$RepresentationID$/init.mp4
        @duration: 3982
        @startNumber: 0
        @timescale: 1000
      representation
        @id: 1_stereo_128000
        @bandwidth: 128000
        @audioSamplingRate: 48000
```

Figure 10 MPD_example Content

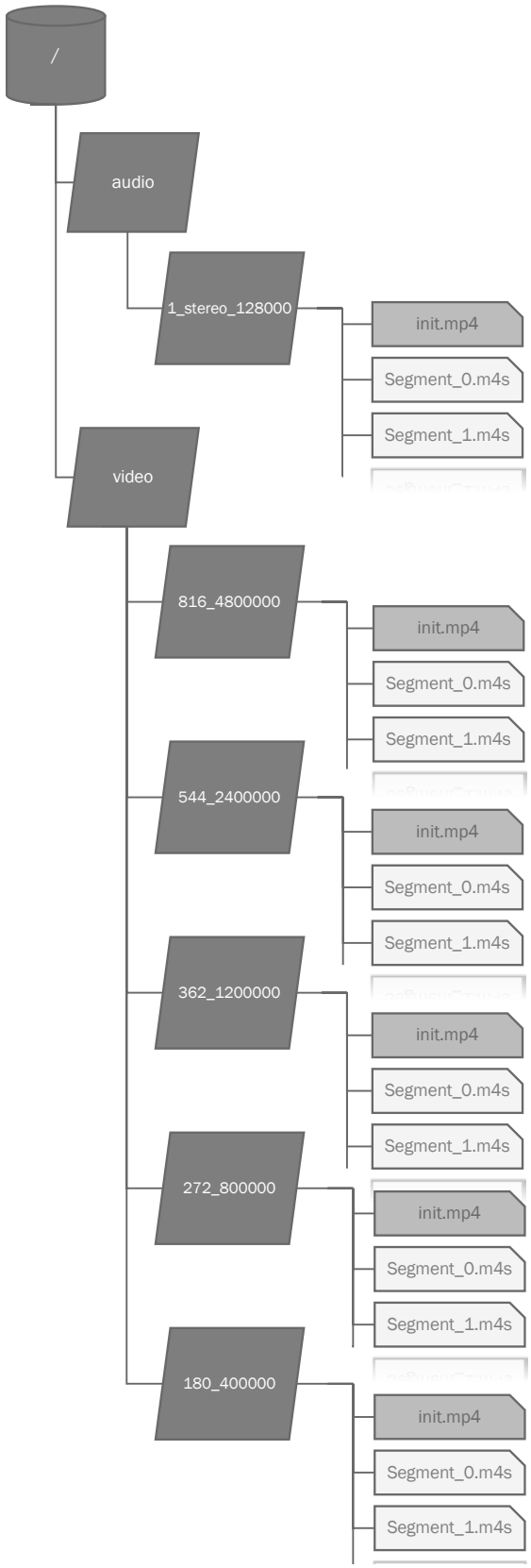

Figure 11 Server Content

Once proven that, we can generate content and play correctly on the client without modifying, proceeded to implement the code locally, this way if able to, following, introduce modifications in order to add a security module that is what concerns us.

3.2.2. Local implementation

Firstly, to execute this client we have to setup a server-side application, which is implemented with *node.js*.

Node.js is an open-source, cross-platform JavaScript runtime environment for developing a diverse variety of tools and applications. Although *node.js* is not a JavaScript framework, many of its basic modules are written in JavaScript, what gives us the ease-of-use modules already created and make them serve for our goal and we can make modifications easily in JavaScript. The runtime environment interprets JavaScript using Google's V8 JavaScript engine.

Thanks to this ecosystem that provides us with *node.js*, through its repository of modules called *npm*, first we can make use of one of them in particular, *Grunt*, a JavaScript task runner. An automation tool for performing repetitive tasks like compilation, unit testing and linting (analyse code for potential errors), what will allow us to modify our client and filter errors in a first instance.

Once we have implemented and tested the solution locally, we shall proceed to the design of the Security Module

3.3. Secure module implementation

First of all, it has made a first approximation to the security protocols that allows us to DASH.

As we have already seen we can basically use two security protocols that comply with the standard, focused in two radically different ways. In the first place, the use of the data is controlled by encryption, and the second controls the access to the content through tokens.

From this point, we have created a *git* repository with the client base, which will allow us to make modifications on the code in a safe and orderly manner.

3.3.1. Access control

With regard to the first method of protection, the access control by testimonies, it is basically to manage the server so that it does not respond to the requests not to be containing a valid witness. In addition, as we have already seen in the standard dash, has to indicate to the client the requirement to obtain and submit a token with the request of the content, and this has to interpret, manage and add the corresponding testimony. There are three defined protocols, which differ essentially in the way to deliver testimony to the client. The first two making use of exchange that is already produced files using HTTP, and a third that depends on an external protocol. Due to the simplicity of the protocol on which is used only the MPD to indicate and transfer the token, it has chosen to implement the second one.

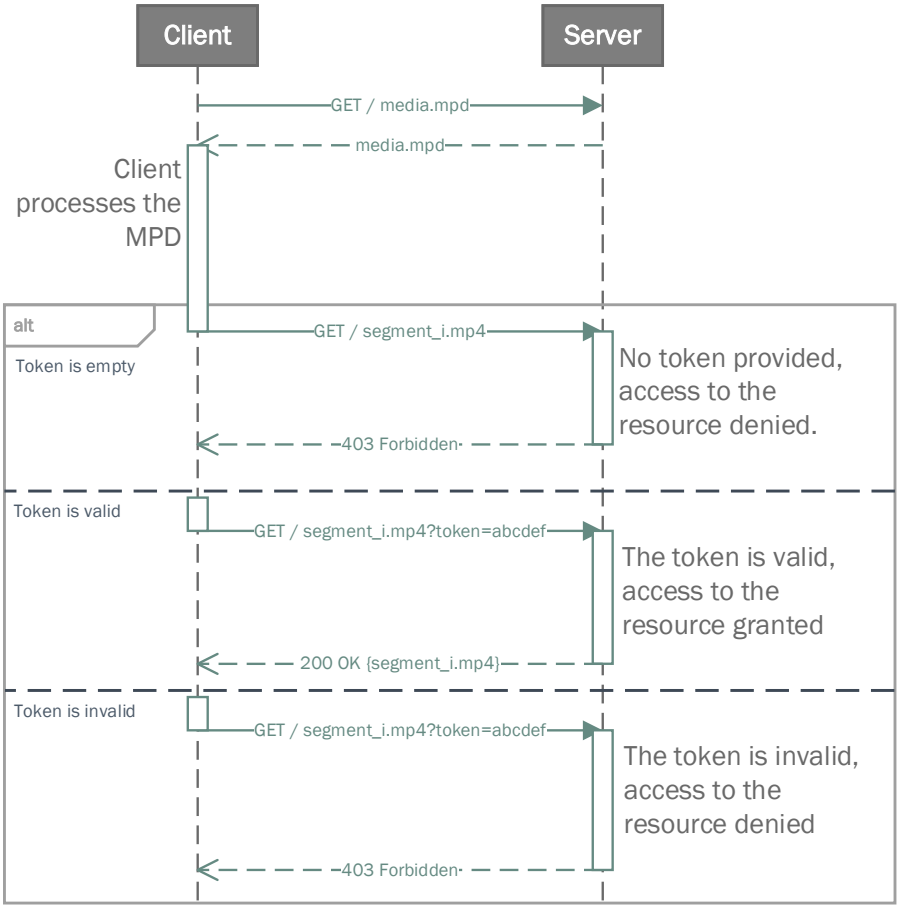

Figure 12 Token Data Exchange

Analysing this system, it has been seen that is in phases of deployment at this client, in addition in a very early phase, which does not allow us to see any functionality and that could interfere in our implementation without making an analysis of way too exhaustive of the functions and system implemented. In addition, the encrypting option in a principle prevents us from having to setup a specific server that can handle the tokens, it will simply upload the content that will be modified, being able to move all of the processing tasks to the client and allowing to use any HTTP server. Thus we will focus on this option, the security implementation from the encryption of information.

3.3.2. Encryption

In the first place, with regard to the compatibility of the standard DASH to implement encryption and DRM in the content and therefore include the necessary information in the MPD, we find that there need too many resources in terms of services that request and manage keys and the DRM for its successful implementation.

Due to this has been considered to implement a simple encryption point-to-point by adding two modules, one on the server and another on the client, so that served as a proxy, encrypting and decrypting the bytes that is demanding the client, so that it is completely transparent to the rest of the client.

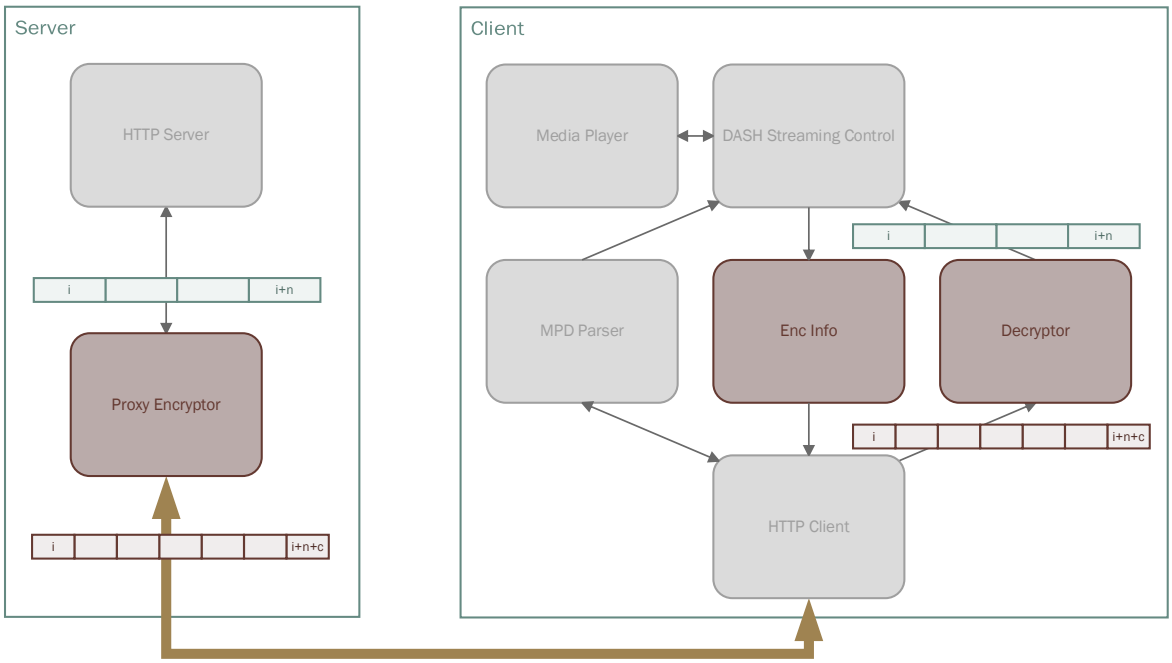

Figure 13 Proxy Implementation Block Diagram

At the time of performing this implementation, we find two main problems to be solved. In the case of demand an entire file, this encryption can be done offline, i.e. as the file only depends on himself we can encrypt and decrypt as a whole. However, the standard dash allows obtaining the content using the sequences of bytes, in this case the client is demanding specific byte sequences, which can be determined by the initial byte and the number of bytes. However, the encryption creates a dependency of the bytes above and in addition is not maintained this index of bytes.

The first thing we need is to encrypt only the bytes requested from the initial byte, because the encryption is performed by blocks and the result may not depend on previous bytes which are not required to send. And in the second place, it is necessary to keep a correspondence between the number of bytes requested without encrypt and delivered to encrypt.

Therefore, given the complexity of this configuration has sought ways to avoid this excess of generation and processing of information. So it has to be able to encrypt in advance, i.e., have been generating segments which the client demanded complete, choosing to perform the segmentation and generation of MPD, segments encoded in MP4 files (.m4s) or MPEG-2 (.ts) independent, which the client demand complete and we can encrypt independently and so the client can decrypt in block without any additional information that the corresponding key.

In regard to the management of key, has been chosen to send the encrypted key information within the MPD, which there is a field in the standard that allows us to add this data as we have already seen, and so maintain a correspondence between the MPD and the encrypted content, i.e. we can encrypt as we want the content and indicate simply in the MPD, and in addition an independence of encryption with the side of the client.

In addition, this implementation does not need to add any complexity to the media server, which simply has to upload the contents of an encrypted form and its MPD with the necessary information.

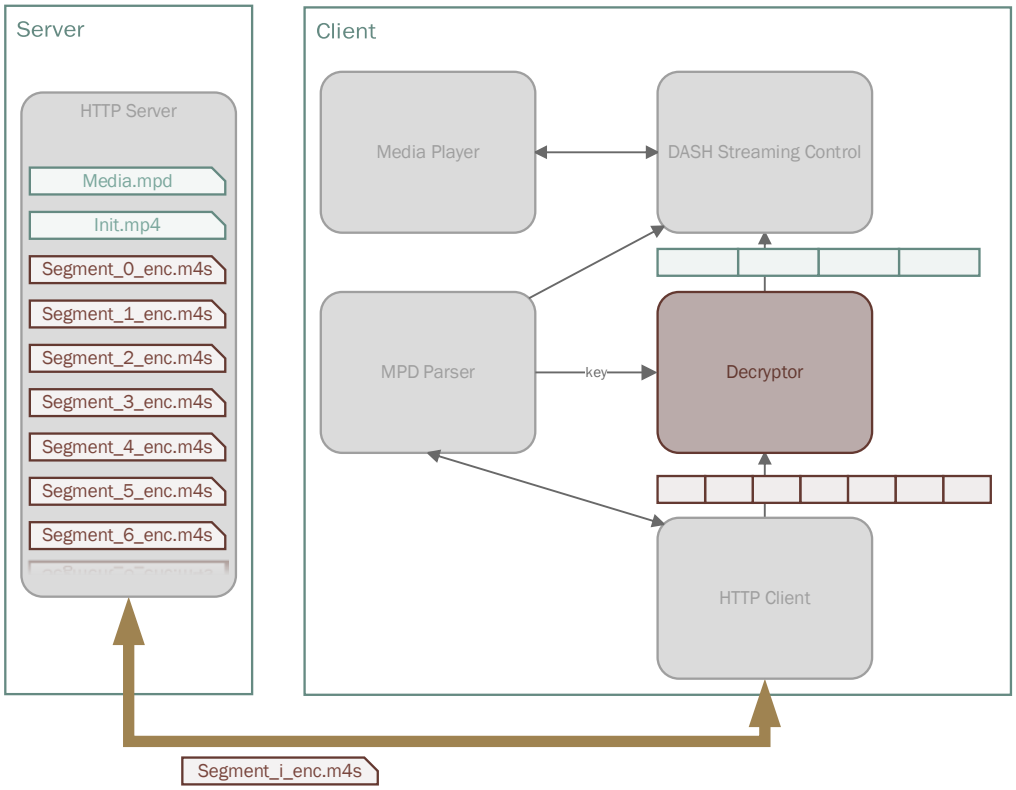

Figure 14 Offline Implementation Block Diagram

First it has proceeded to perform this deployment with a key already predefined and symmetric encryption AES/CBC to test in the first place the correct decryption and playing the content.

To do this it has implement a small program in Java to enable us to encrypt the necessary files. As we have said, it has been decided to use first an AES/CBC encryption, which to be a symmetric encryption by blocks that gives us support on both sides, server and client, and it gives us a good first approximation since we will have to manage the keys and initial vectors own other encryption modes of the same type but perhaps more advanced.

Basically, for this we have to take care that the client can decrypt, i.e., we have mainly to encrypt each file integer and separately. In conclusion, the specifications of our cryptography shall be the following:

- First, with regard to the process of encrypting individual files it be used padding, since we do not have a defined size, in particular PKCS5 padding, which in our case is indifferent to PKCS7 due to the size of blocks that we handle.
- Second, thus due to the size of blocks of encryption (16 bytes per block with AES) it has decided to read, process and write to files in buffers of size 16, since it is the size of the array that can process (encrypt) at the same time.
- Third, as the files to encrypt, we want to protect the content, we do not need to encrypt the MPD or media files that correspond to the initialisation of the segments, because only contributing with metadata, and where is contained the information is in the rest of the files. In addition, it also maintains a certain sending information to the client, which we can take advantage of to contain information of interest to other functions.

Once we have obtained the encrypted content, it has been proceeded to check the decryption of the client side of an artificial way, outside the usual execution of this. To do this it has been implemented only the appropriate decrypting function in JavaScript, which has to fit in with our needs, i.e. that has a mechanism to decrypt bytes, since the most usual in this programming language is to handle strings of simple and easy manner. In addition, have implemented management the PKCS 5 padding used in the encryption.

Among all implementations, the option that is best suited to the above characteristics described is the use of a package *npm*, *aes-cross*, whose main goal is the compatibility/versatility with other codes.

Once we have verified that the resulting information corresponds with the original file, it has been to create a module that includes all the functions to implement the decryption, the decoding of the data received and encoding of the data for their later processing, obtaining and managing the keys and management according to the type of content, encrypted or not.

The first step is to intercept the streaming of data received, for this it has been introduced and implemented a feature that in case that it is required to decrypt call a function to handle this process, and otherwise does not alter the procedure in clear.

First, it will be introduced the role of decryption previously implemented and tested, which now need of decoding and encoding, because the bytes are obtained from a response to an HTTP request ArrayBuffer and also it had to return to deliver encoded so that it can be processed later for the reproduction of the contents. This transcoding is done through different *npm* packages as it gets various function already well implemented.

Once tested the module forcing decrypt the corresponding content of artificial way, we will proceed to implement the function which discerns if this content requires decryption and in which files or requests. To this, it has been to obtain the information the MPD, specifically we use the *essential property* field, which the standard gives us as a mechanism to enter additional information required, but is not well defined its implementation in this case, thus subsequently we will proceed to define a structure. Which we also serve to define and to obtain the encryption method and the data needed to get the keys and decrypt properly. This structure may contain the following fields.

Name	**Use**
mode	Defines the encryption mode
key	Key string base64 encoded
iv	IV string base64 encoded
decryptInRequests	Files to apply decryption

Table 5 Data Exchange to Implement Offline Model

Once defined the corresponding fields, has been validated the compliance with the standard. This is done with an online tool, what allows us to check the MPD compliance using the XSD provided by the standard.

Finally, the system has been tested with the sample content, checking that maintains the standard and its functionality, allows the reproduction of original content unencrypted to

any user, prevents the access to the specified data to users do not authenticated or not allowed and users with the required permissions can play the content allowed.

Finally, we get a module that is running in parallel with the loading MPD module and the loading fragment module. In the first place implements the interception and parsing of all the information of the MPD corresponding to the security module, decoding and storing the information, interacting with loader MPD module. Secondly, it analyses the requests of the content through the interaction with the loader fragments module, comparing with the information obtained previously, and if necessary it decodes the data, choose the suitable decryption method, transforms the data to maintain symmetry with the encryption, extracts the information necessary for the MPD and the user to subsequently decrypt, re-encode the content to be able to reproduce it and return it to the player.

Also it is important to mention that with such checking in a simple way the client, it has implemented a Java application that allows us to generate all the encrypted content automatically, getting all the needed encrypted media content, its corresponding MPD and keys, from a multimedia content and according to specific encryption parameters.

4. Results

This project seeks to obtain a system that can manage security by implementing the DASH protocol. This translates into that this module has to comply with the protocol, i.e. it should be able to play all the content that meets this standard. It must also implement the basic functions defined in the standard. In our case, not alter the correct operation of the functions already implemented. And finally it must be able to access and play content allowed to specific users, as well as to turn to prevent access to unauthorized users. We also have to take into account that we focus in on-demand content. Taking this into account, we have done corresponding tests by trying to reproduce different types of media and MPD contents and we can conclude with the following:

- The client allows the correct visualization of the content without protection to any user.
- The client is unable to play encrypted content without proper authorization (key).
- The client plays the entire contents with the correct key.

- The client is able to change between the segments correctly encrypted.
- The client is not able to manage encrypted content that does not come out of requests of entire files i.e. not segmented multimedia.

- The client is capable of processing and displaying the metadata regardless of whether the content is encrypted.
- The server does not manage keys.
- The server is agnostic to encrypted content.

- Encryption not introduce delay due to be offline.
- The decryption introduces a delay negligible at the beginning and invaluable in the following jumps of segments.

---

5. Budget

In order to develop a budget for the project, several premises and assumptions are detained:

- The project has had a duration of 25 weeks, 20h of work per week.
- The hourly wage is 8€ taking into account an internship contract.
- The project is supervised by two supervisors, dedicating one hour a week.
- The hourly wage of a supervisor is 50€.
- The laptop, table, chairs, furniture and equipment has a cost of 1000€.
- The workplace, rental of space, light supply, connection to the internet and other facilities is 120€ per month.
- The server used to host our service and perform tests has a cost per month of 15€.

Concept	Cost	Total
Junior engineer	8 €/h	4000 €
Senior engineer	50 €/h	1250 €
Senior engineer	50 €/h	1250 €
Equipment	1000 €	1000 €
Workplace	120 €/month	750 €
Server	15 €/month	90 €
	Total	**8340 €**

Table 6 Budget

6. Conclusions and future development:

As a conclusion, due to the standard allows the reproduction of a variety of multimedia sources, there is still give support and define the behavior with another type of content, as is the case of the contents without segmented into files or live broadcast.

As well as, there are also other methods to implement as we have seen, which have different characteristics and handle different purposes, due also to the versatility of the standard and therefore to its use.

In conclusion, we have a relatively new standard, the community are proposing improvements, redefinitions and implementation of the standard constantly, and we have to follow this changes.

Bibliography:

bitmobin GmbH. (2015, December 10). *Libdash Bitdash Whitepaper.* Retrieved from bitmobin: https://github.com/bitmovin/libdash/blob/master/docs/libdash-bitdash_whitepaper_v2.docx

DASH Industry Forum. (2016, July 13). *DASH-IF Implementation Guidelines: Content Protection Information Exchange Format (CPIX).* Retrieved from DASH Industry Forum: http://dashif.org/wp-content/uploads/2016/07/DASH-IF-CPIX-v1.91.pdf

DASH Industry Forum. (2016, July 20). *DASH-IF Implementation Guidelines: Token-based Access Control for DASH (TAC).* Retrieved from DASH Industry Forum: http://dashif.org/wp-content/uploads/2016/07/DASH-TAC-v0.9.pdf

DASH Industry Forum. (2016, June 12). *Guidelines for Implementation: DASH-IF Interoperability Points.* Retrieved from DASH Industry Forum: http://dashif.org/wp-content/uploads/2016/06/DASH-IF-IOP-v3.3.pdf

Delgado, J. (2015, September). Multimedia Content Transmission. Barcelona, Barcelona, Spain.

Delgado, J. (2015, September). XML. Barcelona, Barcelona, Spain.

ISO. (2014, April 14). *Information technology — Dynamic adaptive streaming over HTTP (DASH) — Part 3: Implementation guidelines.* Retrieved from MPEG The Moving Picture Experts Group: http://mpeg.chiariglione.org/sites/default/files/files/standards/parts/docs/w14865-v2-w14865.zip

MPEG. (2016, October 20). *Format Independent Segment encryption and authentication*. Retrieved from MPEG The Moving Picture Experts Group: http://mpeg.chiariglione.org/standards/mpeg-dash/format-independent-segment-encryption-and-authentication

Node.js. (2016, June 25). *Anatomy of an HTTP Transaction*. Retrieved from Node.js: https://nodejs.org/en/docs/guides/anatomy-of-an-http-transaction/

Sodagar, I. (2012, April). *White paper on MPEG-DASH Standard.* Retrieved from MPEG The Moving Picture Experts Group: http://mpeg.chiariglione.org/sites/default/files/files/standards/docs/w13533_0.zip

Glossary

AES
Advanced Encryption Standard 27
API
Application Programming Interface 21
CBC
Cipher Block Chaining 27
DASH
Dynamic Adaptive Streaming over HTTP ... 10, 11, 14, 15, 16, 17, 19, 21, 24, 25, 30, 33
DRM
Digital Rights Management 16, 17, 18, 25
GOP
Group of Pictures 15
HTTP
Hypertext Transfer Protocol 14, 15, 17, 21, 22, 24, 28
IEC
International Electrotechnical Commission 19
ISO
International Organization for Standardization 15, 19
IV
Initialization Vector 28
MPD
Media Presentation Description 14, 15, 16, 17, 18, 19, 20, 21, 22, 24, 25, 26, 27, 28, 29
MPEG
Moving Picture Experts Group 16, 19, 26, 33
PKCS
Public Key Cryptography Standards 27, 28
XML
Extensible Markup Language 16, 17, 19, 33
XSD
XML Schema Definition 28