DESIGN AND IMPLEMENTATION OF A SECURE INTERFACE BETWEEN THE OPEN CONTENT DECRYPTION MODULE AND HARDWARE-ASSISTED DRM

MASTER THESIS

by

ROHAN KRISHNAMURTHY

SUBMITTED IN PARTIAL FULFILLMENT
OF THE
REQUIREMENTS FOR THE DEGREE OF

MASTER OF ENGINEERING

accepted by:

Prof. Dr.-Ing Kira Kastell
First Examiner
FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Prof. Dr.-Ing Sven Kuhn
Second Examiner
FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Dipl.-Inform. Stefan Pham,
Supervisor
FRAUNHOFER INSTITUTE FOKUS

Frankfurt am Main

28 February 2017

FOKUS Institute
Kaiserin-Augusta-Allee 31
10589 Berlin

This dissertation is originated in cooperation with the
Fraunhofer Institute for Open Communication Systems (FOKUS).

This thesis would not have been possible without the guidance and the help of several individuals who in one way or the other contributed and extended their valuable assistance in the preparation and completion of this study.

First and foremost, I express my deep sense of gratitude to Frankfurt University of Applied Sciences and Fraunhofer FOKUS, Berlin that has provided me an opportunity in fulfilling my most cherished desire of pursuing post-graduation.

I would like to thank Prof. Dr.-Ing Kira Kastell and Prof. Dr.-Ing Sven Kuhn from the department of Computer Science, Frankfurt University of Applied Sciences, for accepting my request to guide my thesis work. They were very supportive and it was a pleasure working with them.

I am very grateful to Stefan Pham at the Fraunhofer Institute FOKUS for giving me the opportunity to carry out the state of the art research in this field. He encouraged me to tackle challenges which, at first, seemed unassailable. He also involved in discussions of ideas and thoughts with me. He was supportive throughout the thesis. He gave valuable advice on working in on a scientific project and also helped me in structuring of the thesis report.
This thesis would not have been possible without his support.

I would also like to thank all my buddy friends and colleagues who supported me and participated in the surveys and made this thesis possible.

I very much thank for the support and encouragement from my parents, my grandfather, Sreerama Moodithaya, and all my family members to pursue my higher studies in Germany.

Thank you very much.

ABSTRACT

High resolution motion picture such as High-Dynamic Range (HDR) and Ultra HD 4K/8K are becoming widely popular in streaming community. The recent compendium released by Movielabs [1], beacons the specification standards for enhanced content protection limiting the impact of hacks to DRM schemes. The interaction between the web application and content protection system is enabled by W3C EME (Encrypted Media Extensions), which provides an API that allows the playback of encrypted content. The DRM functionality is achieved by a Content Decryption Module (CDM) in the HTML5 based web browser to enable DRM specific decryption and playback of the protected media content.

Currently, the DRM schemes such as Google's Widevine, Microsoft's PlayReady, Apple's Fairplay and Adobe's Primetime are integrated with Web Browsers as proprietary CDMs. The coexistence of such various schemes brings up the challenge to build a platform independent CDM. To meet this challenge, Fraunhofer FOKUS has open sourced the Open Content Decryption Module (OCDM) on GitHub. The OCDM is developed with a secure media pipeline along with the hardware assisted DRM such as Microsoft's Media Foundation which verifies the license validity and transfers the decrypted digital data to the video player. Hardware-assisted DRM can be achieved by the Microsoft Content Decryption Module interface (CDMi) which is an interface to the actual DRM and mirrors the actual EME interface. For example, Smart TV or set-top-boxes which incorporate the PlayReady Device Porting kit will make use of PlayReady CDMi to project the DRM functionality into the browser. The security of playback content in such embedded devices can be realized by integrating Content Decryption Module and CDMi with a Trusted Execution Environment (TEE).

The question as to how to construct an interface to build the pipeline between the OCDM and the hardware-assisted DRM in a secure way to be implemented on an open source browser according to the W3C EME is the basis for this thesis work.

Work items:

- Enhance the implementation of Open Content Decryption Module
    - Design a secure system architecture for OCDM and the CDMi to be implemented using TrustZone/TEE framework.
    - Be fully compliant with W3C EME recommendation and enable enhanced security for encrypted samples with the support of, e.g. key-rotation mechanism, multiple sessions, etc.
- Chromium open source browser
    - Evaluation of Pepper Plug-in API (PPAPI) sandbox security

    - Integration of PlayReady Device Porting Kit on a TrustZone/TEE using a secure media path via CDMi

- Nightly open source – Mozilla Firefox browser
    - Evaluation of Gecko Media Plug-in (GMP) sandbox security
    - Integration of Microsoft's PlayReady between Gecko Media Plug-in and the Media Foundation API via OCDM

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

There is no such thing as perfect security, only varying levels of insecurity.

– Salman Rushdie

Tele-Vision by 2020 would mostly be about personalized technology, on-demand streams, individualized content distribution, ad-free broadcasting and everyone connected to a Smart TV. People are consuming videos with a better viewing experience that has accelerated video consumption. Live streaming, on-demand streaming and Over-the-top (OTT) has given the consumers a new face of video generation, distribution and consumption methods. Several entertainment studios, one amongst many is the NHK Japan research lab, is stepping it's feet for enabling the audience with a viewing experience for 8K Ultra HD which is 16 times the resolution of a full HD. NHK is planning to deploy the 8K at the Tokyo Olympics in 2020.

Brace yourself!!

Broadcasting of such live streaming videos to every audience all over the world is one of the key technologies that is playing a big role in strategic business model for the content distributors and content seekers. The creative live streaming video initiative has emerged as the medium of choice for business-to-customer (B2C) and business-to-business (B2B) communication. According to Cisco's June 2016 Visual Networking Index report, the streaming video would see an increase to 82% of the internet traffic by 2020.

With the advancement in the multimedia industry mainly concerning the available product and services across all platforms and multiple devices, there is a need to enhance the content protection so as to alleviate piracy threats that occur during the exchange of digital data. The amount of involved risks in piracy gets higher while dealing with adaptive streaming technologies such as MPEG-DASH with Common Encryption (CENC). And this is more prone when the secure media pipeline has been compromised. Browsers can play DRM-protected content, and the most common way is to enable DRM-enabled versions of Adobe Flash or Microsoft Silverlight. These components were previously available as browser plug-ins. They (and their underlying media platforms) used to standardize media channels and provide plug-in playback for free for end users.

The roots of Flash plug-in are known to have emerged for almost two decades. But of recent times, it has been glued with major security vulnerabilities[1]. The Chrome browser has already dropped the support of Flash support and by the end of 2017, except for few online games, the flash plug-in is believed to bite the dust giving rise to HTML5. With this trend set-up, next-gen video formats such as h.265 (also known as HEVC) and VP9 would significantly improve the performance of online video streaming world. The current technology involving HDR (High Dynamic Range) and HFR (High Frame Rate) that are supported on 4K TV's would be partially enabled because of HEVC and other growing standards. As the internet speeds increase and new ways of compression schemes, H.265 is expected to deliver 4K video at 15-20 Mbps (which is most commonly available at home broadband connections).

Plug-ins can restrict web content-compatible devices that support HTML extensions, which allow browsers to directly support rich functionality and reduce the need for plug-in components. This allows the browser to play media files directly without the need for any media player plug-ins. The new specification now defines HTML features that allow adaptive streaming of DRM-protected media.

1.1 Motivation

As the content owners' demand to deliver the premium protected content to end customers, the service providers implements the content protection systems to fulfill these requirements. And on the other end, the end customer demands the protected content to be portable across different platforms. This project studies the hardware based DRM to meet both content owners and customer's demands in a secure way by developing an open source content decryption module.

Research questions: Consumption of the digital content over different platforms and at the same time protecting the content by using hardware based DRM can discussed over 3 main technical aspects:

> *Open Source CDM*: How to design and implement a secure interface between OCDM[2] and hardware-assisted DRM?
>
> *Portability*: How to protect the digital content from illegal content proliferation while still managing to distribute the content across different devices on different platforms?

1https://www.cnet.com/news/adobe-rushes-out-emergency-update-for-critical-flash-security-flaw/
2https://github.com/fraunhoferfokus/open-content-decryption-module

Implementation: How can a DRM system be implemented so as to satisfy the end customer’s expectation and to provide high end security against the possible third-party attacks?

The thesis unfolds these technical aspects by developing and implementing the secure content decryption module using hardware based DRM on Mozilla Firefox- Nightly browser and studies the design of TEE architecture for software based DRM on Chromium project.

1.2 Overview

The first phase of the project from chapter 2-3 studies the available existing DRM schemes and the standards to integrate the DRM schemes to get an overview of the current market on protecting the digital content. This will provide a good understanding of the research questions and the possible ways to consider the demands of the content owners and end customers.

Chapter 4 provides the architectural view of TEE to enhance the security vulnerabilities.

Chapter 5 is the implementation phase showing the workflow of the Open Content Decryption Module on open source browsers such as Chromium and Mozilla Nightly.

- Microsoft’s CDMi on top of the existing Open source Content Decryption Module to realize the software based DRM using Chromium project. The Chromium uses the PPAPI plug-in along with ffmpeg decoder to deliver the playback of the PlayReady protected content. This approach is realized without SANDBOX security and hence the project highlights the possibility of using TEE to secure the crypto keys as a Trusted Application (TA) and beacons the standard specifications set by Global Platform.
- Followed by, playback of protected media using Media Foundation media player which acquires the DRM license using a secure media pipeline.
- The Microsoft’s PlayReady DRM with Media Foundation to realize the secure distribution of digital content on Mozilla Firefox-Nightly. The Firefox browser makes use of the GMP plug-in and Media foundation to decode the digital samples and hence provide the end-to-end security.

The third phase of the project from Chapter 6 studies the security aspects that were considered during the implementation phase.

Chapter 7 draws the conclusion and also provides an insight to the possible enhancements.

Chapter 8-10 consists of the bibliography of references, appendices and the glossary index.

2. DRM

DRM schemes enable the control of the distribution of digital content. Content owners and the distributors select the desired schemes to protect their content and the vendors could possibly support more than one DRM scheme, so as to distribute their content on various platforms.

Till this date, the DRM schemes are developed by several organizations that promote the consumption of the DRM protected content across different systems. And these systems augment the acceptance from the consumers of the DRM scheme to consume the protected contents.

OMA DRM: is an acronym for the Open Mobile Alliance was formed in 2002, by major mobile phone manufacturers and carriers, network suppliers for developing open DRM standards to enhance the interoperability among the devices. There exists two versions of OMA DRM; OMA DRM 1.0 which is a basic DRM standard that offers no support against strong protection of the content and OMA DRM 2.0 supports the protection of content using encryption methods. Further, there is an additional extension available known as OMA Secure content exchange (OMA SCE) that enables protected content to be consumed across different mobile devices.

Marlin: Marlin DRM was founded in the year 2005 and was developed by Intertrust, Panasonic, Samsung, Philips and Sony. It provides an end-to-end digital rights toolkit, which consists of the specifications and the Sample Implementations Kits that forms a complete DRM system. The Marlin DRM's are mostly deployed in the smart TV market.

Fairplay: Fairplay was originally developed by Veridisc and adapted by Apple, Inc. It is a built-in component of the Quick time software that is installed across all lines of iOS devices such as iPhone, iPod, iPad, AppleTV.

Widevine: Widevine was designed by company Widevine which was later acquired by Google in 2010. The Widevine DRM provides the capability to license, securely distribute and protect playback of content on any consumer device supported by Google TV, Google Chromecast, Google Play and other Google Products.

PlayReady: Microsoft introduced the PlayReady DRM in the year 2007, widely used by Windows applications. Currently, Netflix has chosen the PlayReady DRM as its primary DRM technology for the secure distribution of premium video contents.

2.1 DRM S/W and H/W

The Block diagram for a typical DRM architecture is as shown in figure-1.

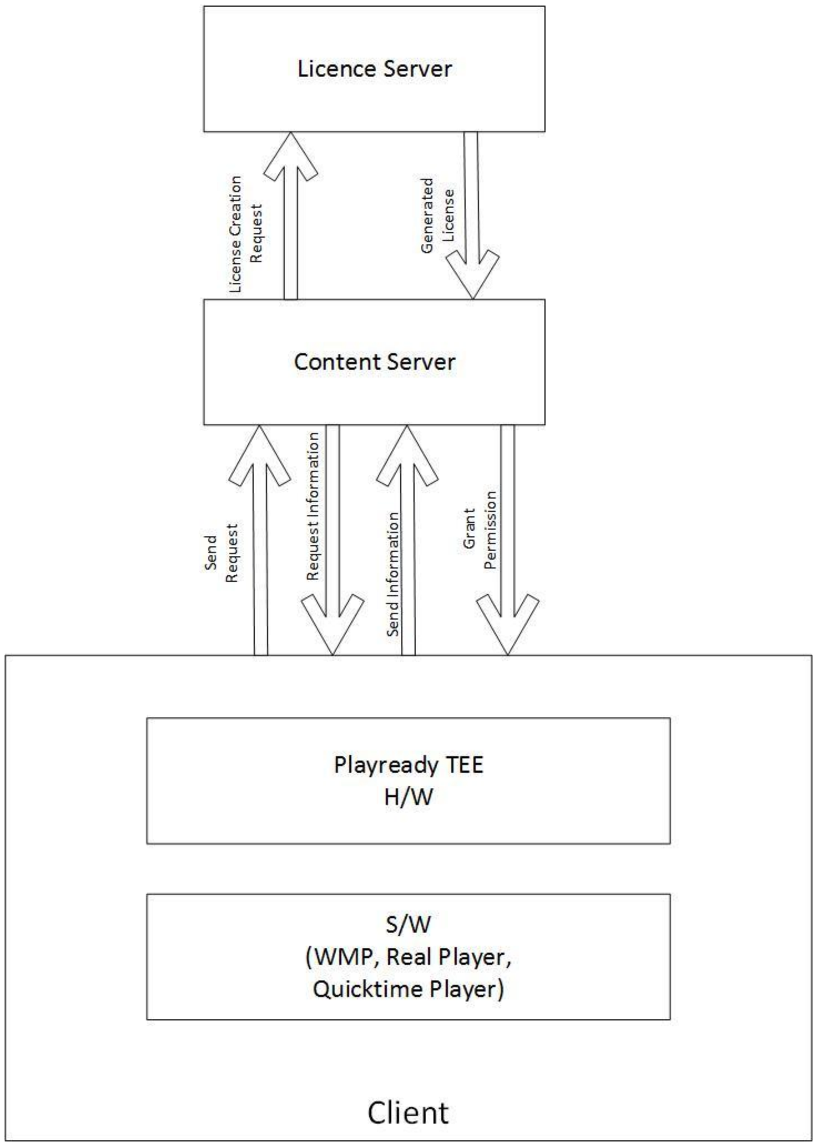


Figure 1: DRM architecture

2.1.1 DRM Software architecture

The media players such windows media player (wmp), real player, quicktime player, receives the encrypted data and its license from the servers. The application performs the necessary operations following the rules of the license. The player plays back the decrypted content and passes it to the user mode (either an external display or any such device). To enable the end-to-end security, the content flow between every link between each component must be secured. The algorithm for encryption the content key is DRM specific.

Content server:

The responsibility of the content server is to encrypt the clear samples by using cryptographic algorithm. Different DRM schemes may deploy different algorithms. These samples are transmitted to the player module via the internet connection. And essentially, the encrypted samples cannot be played by the third party attacker, without its decryption key.

License Server:

The license server maintains the whole database for encryption keys and issues the rights for the contents. The server generates customized licenses for different demands from the customer, e.g. if the customer is willing to pay more, then he will get extra benefits such as, sharing the content, downloading the content to view in offline mode. The license server stores the key pair wherein the *private key* is used for signing the license and the *public key* for verifying the signature.

The player also has the responsibility for handling the device key, secure playback and enforcing the rights. For example: The Windows Media Player has the ability to enforce the Microsoft's PlayReady scheme. The player itself unwraps the content key, decrypts the media file with the key. However, the software based DRM scheme lacks hardware protection against the tampering and obfuscation.

2.1.2 DRM hardware architecture

In comparison to the software architecture, the device secret key must be secured by hardware root of trust without exposing to the software stack. The Microsoft PlayReady implements the Media foundation for decrypting and decoding the samples. The figure-2 shows the flow of the content and license within the distribution system.

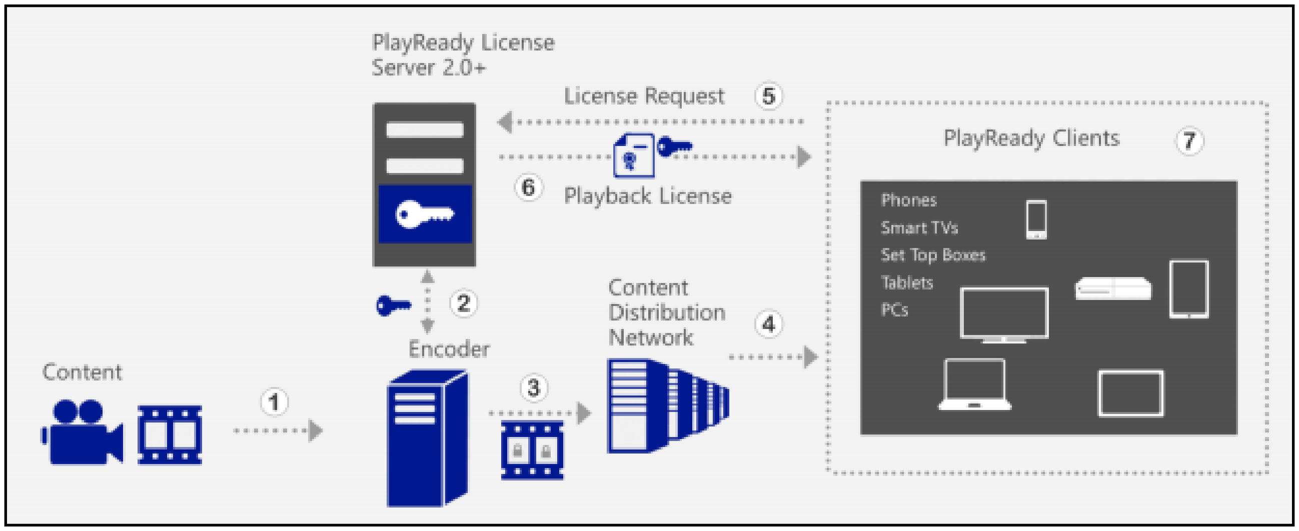


Figure 2: Workflow of the media distribution [2]

1. The media content sends a request to the encoder (for encoding and encryption)
2. PlayReady uses AES-128 CTR encryption algorithm and shares the PlayReady license. Additionally, the encoder also adds the PlayReady header to the media file that enables the PlayReady client to decrypt and acquire the license.
3. The media file is packaged and sent to content distribution Network.
4. In response the CDN sends the media file to the PlayReady Client
5. The client sends a license request to the license server by using the PlayReady header.
6. The license server receives the request and
    - Authenticates the device,
    - Adds the usage policies,
    - Encrypts the license using the client device's public key and
    - Sends the license back to the client.
7. The client decrypts the media file using the private key from the key pair. The client then uses the license to securely decrypt and playback the media file.

Microsoft PlayReady Porting Kit

Microsoft PlayReady Porting Kit: The kit consists of a set of PlayReady APIs, portable source code, test resources, and documentation to a wide variety of system architectures and operating systems. There are no specific hardware or system requirements and has been tested on Linux distributions and Windows OS. The API provides a definitive component for calling PlayReady functions from an application. These functions are basically required to perform functionalities such as initializing a session decrypting and rendering protected media streams. Others functions do exist and can be implemented depending on the features required to be implemented. For example, key rotation and ad insertion for Live TV content and backward compatibility with content that is protected by Windows Media DRM (WMDRM) technology.

***Compliance and robustness**:* The content protection regime defines and enforces set of rules, a chain of trust and establishes remedies for security breaches. The PlayReady compliance and robustness rules are specified in PlayReady license agreements and all PlayReady implementations and products must satisfy the requirements defined in those rules. *Compliance rules* describe how media content may be accessed and shared according to specific rights and restrictions. *Robustness rules* specify protection requirements for each type of media asset and ensure that implementations are designed to protect content in a robust manner. The compliance and robustness rules can be read from the PlayReady website[3] maintained by Microsoft.

[3]https://www.microsoft.com/PlayReady/licensing/compliance/

2.2 Media Foundation

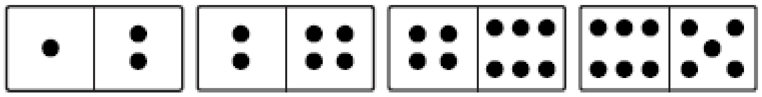
Microsoft Media Foundations is the new media platform in Windows superseding and replacing the Microsoft's Directshow, DirectX media objects and all other previous media technologies. The following sub chapters illustrate the use of MF components and modules to process audio and video data streams.

MF Pipeline architecture

Audio and video streams are compressed using various compression algorithms and are stored in different file formats also known as containers, that simplifies video processing used for different operations. *Pipeline* is a generic term used to design a series of connected components without forming any loop of operations to process the data flowing through them.

Consider a video stream sent over the network as an example. These videos samples are played as and when the data is being received. And to playback these video samples, a program in the form of an application needs to perform a set of operations on the data received to use the data streams from the container packet, then decode (uncompress the samples) the data and finally paint back on to the screen.

Consider the MF components as a set of domino pieces as shown below.



These pieces are mutually connected with each other forming a chain of operations that are performed to process the media streams. To play the video file that was previously considered, would basically require two sets of MF components: one to decode and display the video, one to decode and render the audio. And these two connected components is nothing but a pipeline.

As a basic example, let us consider the steps necessary to playback a video file. The MF pipeline needs to perform a set of operations to playback the video file, such as,

1. Load the file from the disk,
2. Unpack the data streams,
3. Separate the audio and video by their respective codecs,
4. Decode – decompress – audio and video data,
5. Render the uncompressed and decoded data,
    a. Audio data to the speaker
    b. Video data to the display screen

The following figure-3 shows the design of MF pipeline used to play the video stream. Steps 1, 2 and 3 are done by the file source, step 4 is performed by audio and video decoders and step 5 is done by the audio/video renderer's.

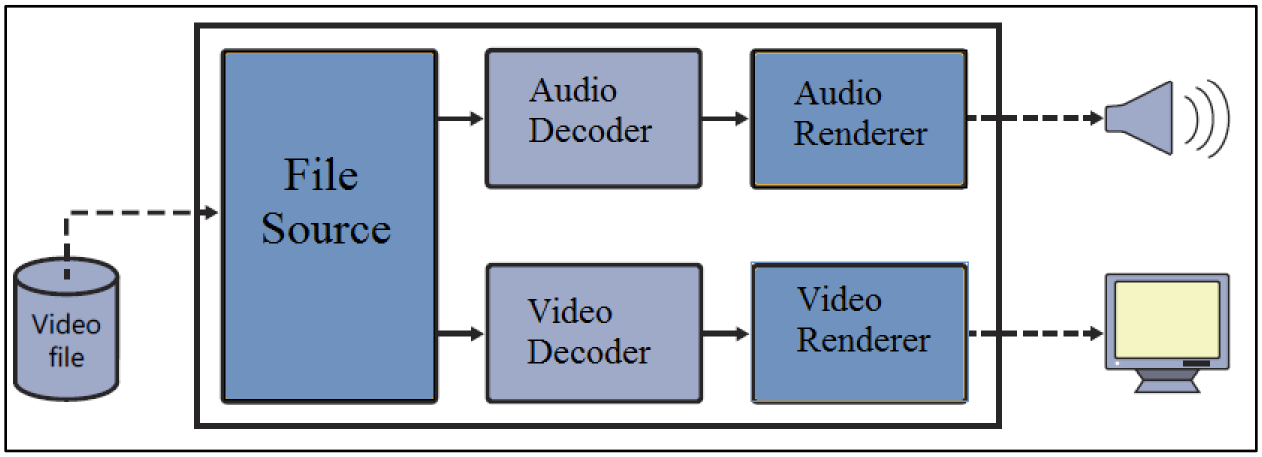


Figure 3: Media Foundation pipeline

The Media Foundation [3] presents two approaches to initialize the pipeline. As shown in the figure-4 below, the first method initializes the pipeline by providing an URL or uploading the media files the disk, thereby having an end-to-end pipeline architecture for handling the media stream. And in the second approach, pipeline is initialized using the push-pull approach. The application pulls the media stream from the source and/or pushes the stream towards the destination.

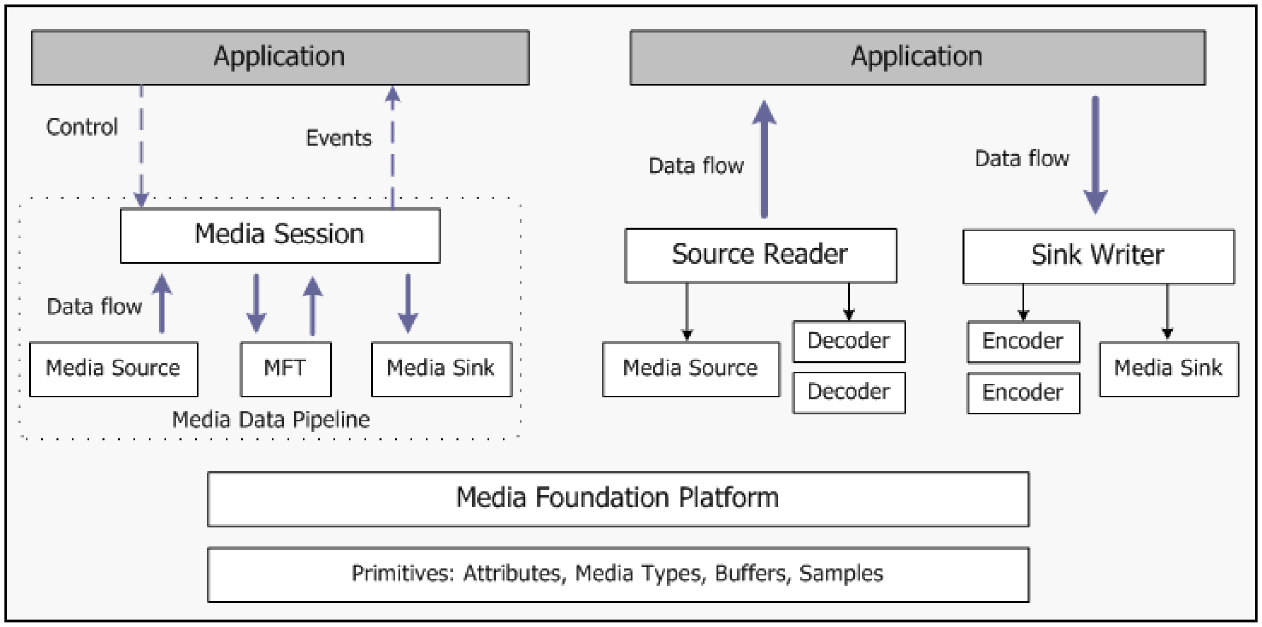


Figure 4: Media Foundation Architecture [4]

The development of such a basic pipeline has been the foundation to implement the hardware assisted DRM for Nightly browser. However, the pipeline is constructed in such a way so as to bridge the data flow from the GMP plug-in from the Nightly browser and the Microsoft Media foundation. The implementation of this scenario can be observed in chapter 5.2.2 with reference to figure-39.

Similarly, the Microsoft media foundation based media player has been implemented in the chapter 5.2.1 to present the media stream flow within the media foundation that acquires the DRM license automatically and enables the playback of the protected content.

The understanding of the Media Foundation API with respect to the EME standards will be studied in chapter 5.2.2 and call flow is shown in the figure-34.

3. STANDARDS FOR INTEGRATING DRM

3.1 HTML5 MSE/EME

The streaming of the digital content over the web had started in the early 90’s facing the challenges such as timely delivery, lossless consumption of large amount of data. The IETF designed a standard Real Time Protocol (RTP) to deliver the video and audio packets with stream session management. However, in today’s world, these managed networks are replaced by Content Delivery Networks (CDN). The media content are now delivered efficiently in large segments over HTTP and is one of the popular approaches to deliver contents that are deployed commercially, for instance, Apple’s HTTP Live Streaming (HLS) [5], Microsoft’s Smooth Streaming [6] and Adobe’s HTTP Dynamic Streaming [7].

However, these streaming platforms use different manifest and segment formats, thereby lacking system interoperability. Hence, there was a need to standardize the HTTP streaming for multimedia content. In 2014, MPEG coined the standard specification termed as MPEG Dynamic Adaptive Streaming over HTTP (MPEG-DASH)[4] to meet the demands of interoperability.

In the current generation of internet video market, big companies like Microsoft, Apple, and Adobe have developed their own native applications to support decryption and to render the premium video content. These native applications were later designed to be the browser extensions that enabled the HTML embedded encrypted content. And with the advent of adaptive bitrates (ABR) streaming paradigms such as Microsoft’s Smooth Streaming and MPEG-DASH, the content distributors needed more secure way to control the playback experience. This led the W3C[5] to develop the specifications for HTML5 known as Media Source Extensions (MSE) [8] and Encrypted Media Extensions (EME) [9].

[4]MPEG-DASH is an adaptive bitrate protocol based on open IETF international standard, to solve the issues related to the distribution of media contents to multiple devices.
http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=65274
[5]World Wide Web Consortium (W3C) is an international standards organization founded in 1994 that develops the technical standards and guidelines for the Web.

MSE enabled the JavaScript to feed media streams to the video element for playback. With this, the web applications are able to implement adaptive delivery protocols, playback of video files and content insertion.

Encrypted Media Extensions (EME) defines an API for HTML5 video element that enables the playback of protected content. The web application could be well designed to playback protected content by any EME DRM.

3.2 MSE/EME architecture

MSE and EME provide the specifications for the W3C HTML5 extension that define new API's for adaptive delivery (MSE) of protected content (EME). Figure-5 below shows the MSE EME architecture.

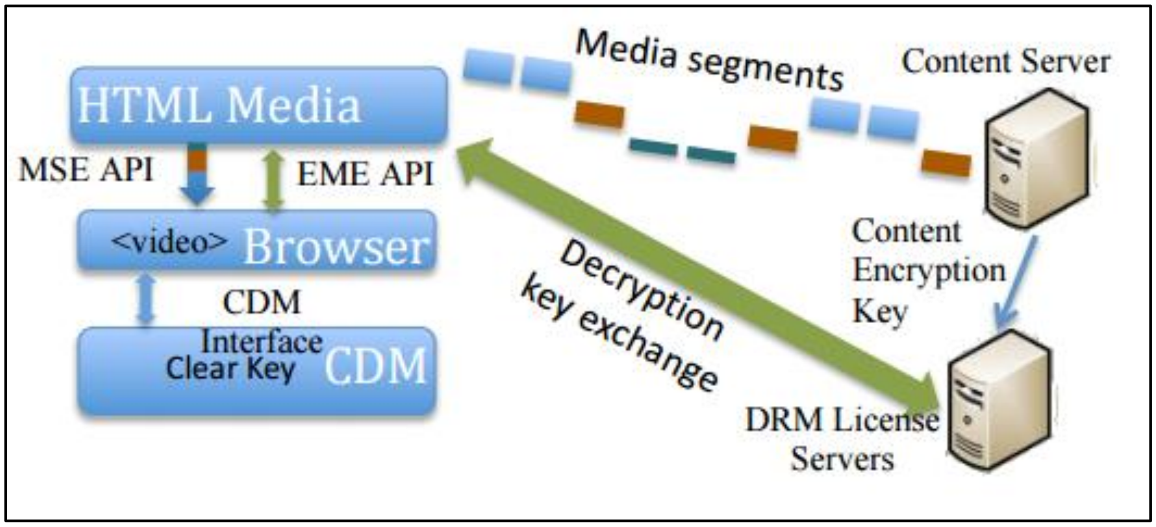


Figure 5: MSE/EME Architecture [10]

The MSE specification defines an API to feed media data from a web page to the HTML5 video/audio element. This is achieved by enabling the JavaScript in the web page to:

- Handle the process of an adaptive media manifest file,
- The media segments are fetched from the manifest file using the URL
- The media segments are appended by the web media player for playback

The EME specification defines an API that enables the web page to playback the protected content by the DRM system using video/audio element. The API enables the web page to:

- Detect the possible attempts on playback of the protected video
- Check for the DRM scheme to playback the content

- Request for the DRM license required for content playback
- Provide the acquired license to decode the protected content.

But MSE/EME together only provides an API to interact with the Content decryption Module, whose functionality could either be:

- Decryption only, the playback is enabled using a media pipeline
- Decryption and decoding, the video frames are passed to the browser
- Decryption and decoding, the video frames are rendered directly in the hardware

3.3 MPEG-DASH CENC

The streaming technology has taken a strong leap towards a client-based adaption logic that requires a manifest description provided by the server and content retrieval by HTTP. The MPEG-DASH standard suffices this growth in streaming technology by embracing proprietary streaming formats into an international standard.

With several DRM's existing in the market, allows the possibility to adopt to various encryption methods to license the files. However, the video providers would need to replicate and adopt the same encryption methodology every time. In order to overcome such ambiguities, Common Encryption (CENC) is defined based on W3C standard (standardized as ISO/IEC23001-7). This scheme specifies a " standard encryption and key mapping methods that can be utilized to enable decryption of the same file using different digital rights management and key management systems" [11]. It also defines a standard format for the metadata necessary to decrypt the protected systems. And hence, the content providers can encrypt the media streams once per codec and consume it with many Key Systems. And the DRM systems provide the right key by using "CENC" key Identifier. Each ISO Base Media File encrypted according to CENC presents a box termed as Protection System Specific Header (PSSH) box, containing DRM information such as licenses in order to distinguish between multiple DRM systems.

4. TEE

TEE, the Trusted Execution Environment is a dedicated area in the heart of the processor of any connected device which ensures the sensitivity of the application by storing processing and protecting in a trusted environment. A TEE offers an environment to execute any authorized software, termed as ‘Trusted Applications’ (TA), enabling the end-to end security by protecting the confidentiality and the access rights against the software attacks in rich Operating System (OS). TEE can be realized as a bridge between the rich OS and the Secure Environment (SE).

Digital content not only requires a high level of functionality with quality features expected by the end-customers, but also a high level of security against piracy, redistribution of copyrighted applications. These security aspects are met by creating an isolated environment from the “normal” environment, also termed as Rich Execution Environment (REE) Global Platform, ‘‘TEE system architecture,’’ 2011 [12]. The TEE is isolated from the Rich OS and protects the assets such as crypto keys and user credentials or any such confidential attributes. A legitimate process in the Rich OS uses a communication channel to access the confidential attributes residing in TEE. This process can send a request to (and receive the response from) the TEE.

4.1 TEE as an environment for protected media playback

The platform on which the TEE is implemented would provide an isolation from the normal OS by enabling the security features for the media content. The secure environment would involve security features such that the decoding of the decrypted content, decompressing the content and rendering the samples towards the normal OS. The platform would also implement a set of security features to protect assets such as distribution policy, authentication of rights, and storage of keys etc. without adversely affecting the system performance.

4.2 Software architecture of a TEE

TEE is enabled in the hardware system by isolation of the operating system and creating the secure boot process independent of the hardware architecture of the device.

The TEE software architecture identifies two different components:

- TA the trusted application that makes use of TEE internal API
- Trusted OS components as required by the TA’s that is accessible by the TEE internal API

The REE also identifies two different components:

- The applications that make use of TEE Client API offered by the TA's
- The Rich OS that provides send the requests to the TEE.

The high level architecture view of the embedded TEE alongside REE is as shown below. The TEE, however, is independent of the hardware architecture of the device itself.

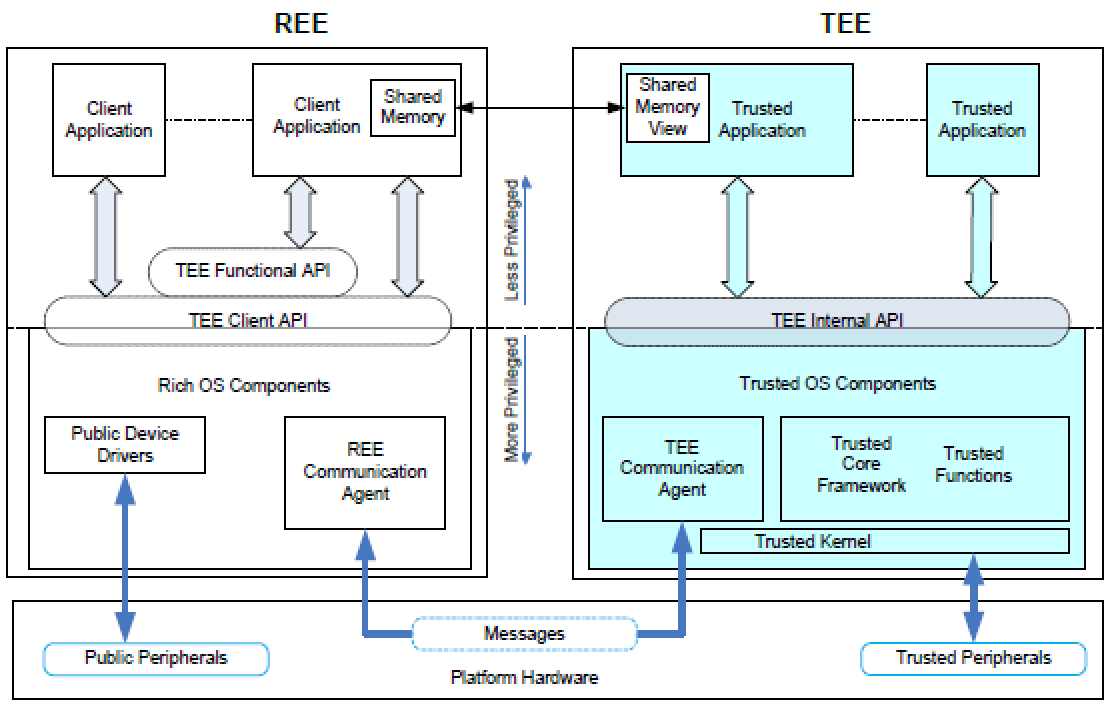


Figure 6: Software architecture of TEE [13]

4.3 Hardware Architecture of TEE

The physical boundary of the TEE is independent of the hardware being implemented on. Logically, The TA's used by the TEE are separated from those applications residing in REE and is as shown in figure-7. More details on the physical components of the TEE architecture can be found in the technical specifications set by Global Platform [13].

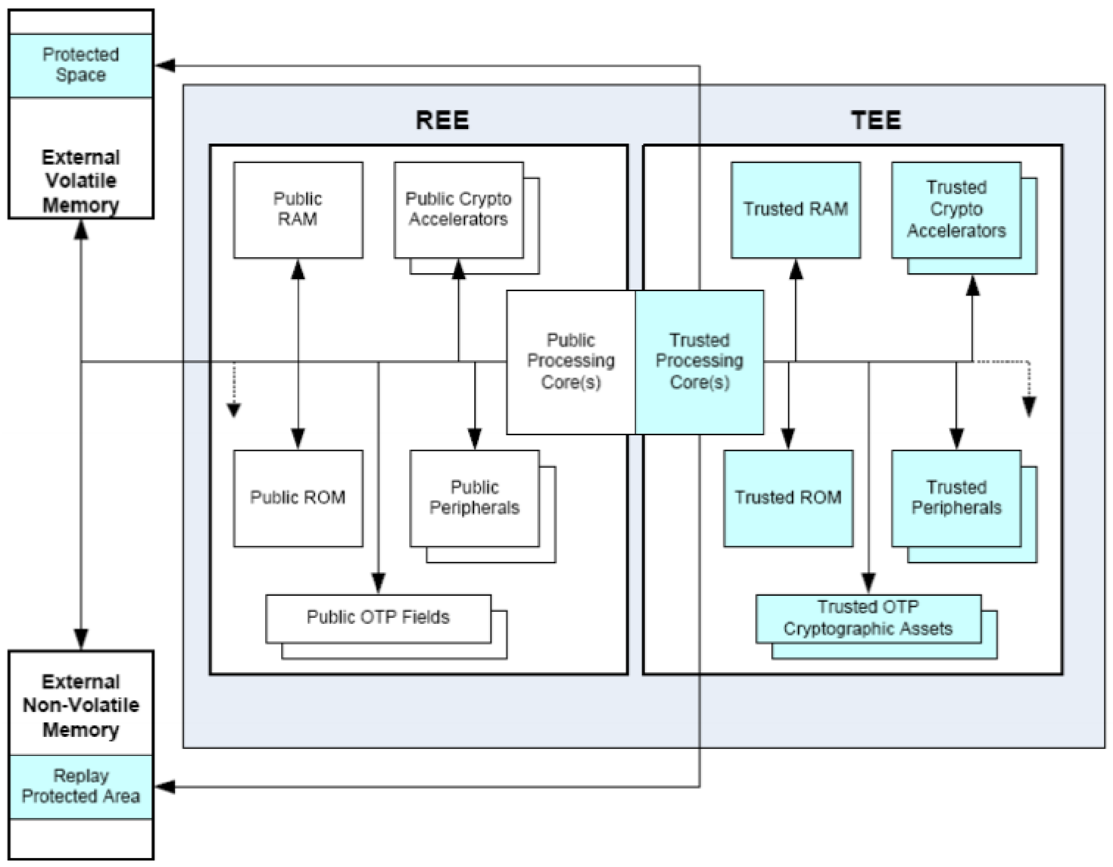

Protected Space
External Volatile Memory
REE
TEE
Public RAM
Public Crypto Accelerators
Trusted RAM
Trusted Crypto Accelerators
Public Processing Core(s)
Trusted Processing Core(s)
Public ROM
Public Peripherals
Trusted ROM
Trusted Peripherals
Public OTP Fields
Trusted OTP Cryptographic Assets
External Non-Volatile Memory
Replay Protected Area

Figure 7: Hardware architecture of TEE [13]

4.4 Protective Shield of TEE

DRM Factors	**TEE principles**
Boot Process	Secure boot process to enable the trusted environment
Code Execution	Hardware-based isolation
DRM keys (Cryptographic schemes)	TEE secure key management
DRM storage rights	TEE secure storage
Secure media playback	Configuration of hardware firewall
Link between normal OS and Secure OS	IP and HDCP configuration executed in TEE

4.5 TEE Standardization: Need

Standardization specifies the common API's for setup and execution of trusted applications across hardware devices. Various industry standards that define services which make use of the TEE functionality include the Trusted Platform Module (TPM) [14]. The Open Mobile Terminal Platform (OMTP) and Global Platform. Global Platform, in particular amongst the other industry standards has identified the necessary technical specifications to develop and publish the secure way to manage multiple applications in the TEE. The TA's that are executed in TEE zone are as per the specification mentioned under TEE internal API [12]. The internal API includes interfaces for cryptographic functionality and secure storage. Global Platform also specifies the standardized implementation method for DRM applications to store and access the license keys, usage policy, or any such authorized rights for the trusted applications. Additionally, the TA's interface with other TA's or component modules, either secure or unsecured (media rendering or scheduling) via various API's such as HTML5 inside the TEE.

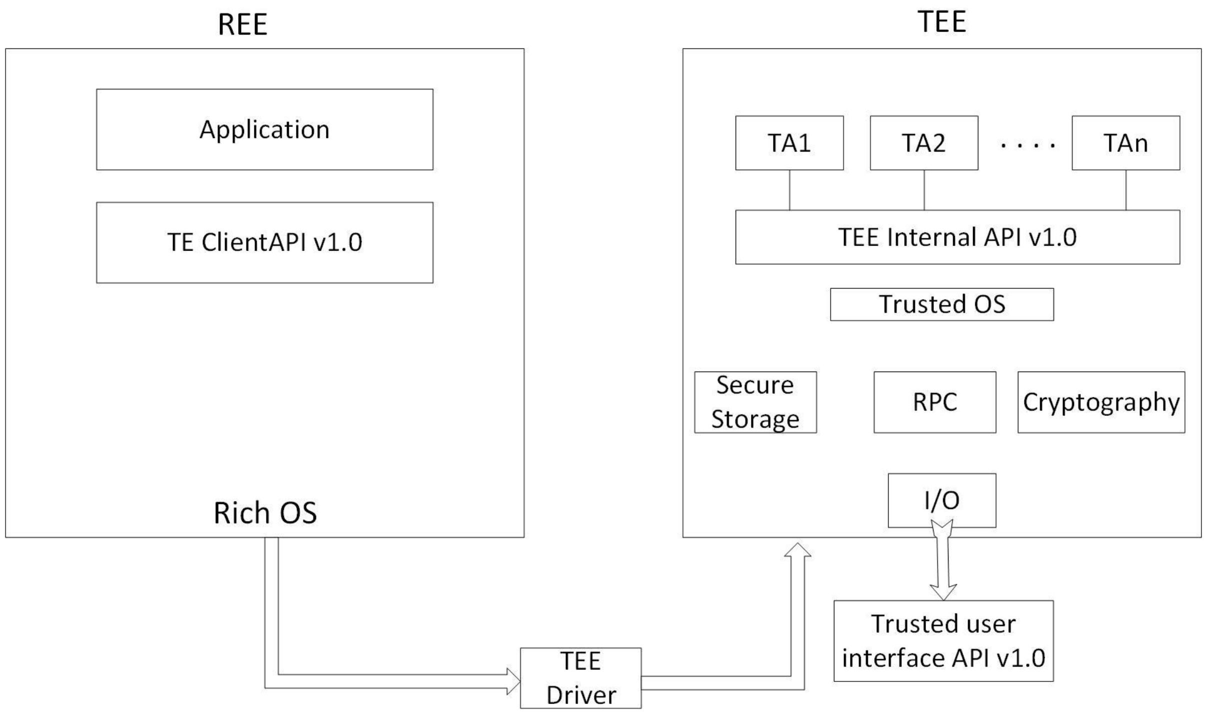


Figure 8: TEE

4.5.1 OPTEE

OP-TEE is an open source developed with collaborative work between STMelectronics and Linaro (Security Working group). OPTEE contains three main API's that comprises of client

API's (optee_client)[6], the TEE driver (optee_linuxdriver)[7], the Trusted OS (optee_os)[8] and the test suite (xtest)[9].

Architecture support: (Both 32 and 64-bit support) developed and tested on ARMv7-An architecture support for ARMv8-A platforms (Allwinners A80, FVP, Juno, HiKey etc).The build process of OPTEE can be in the Github repository.

The build process of OPTEE can be found in the Github repository[10].

4.5.2 Understanding OPTEE-OS codebase:

core/arch/arm/kernel:
Code for the core kernel.

1. abort.c - basic exception handling is defined here, needs to be rewritten considering RISC-V exception handling.
2. cache_helpers.S - cache handling.
3. elf_loader.c : platform specific
4. generic_boot.c: Booting and initialization of normal and secure worlds, Access some co-processor registers.
5. generic_entry_a32.s : handling boot and reboot
6. kern.ld.S : boot initializations, platform specific
7. mutex.c: defines a mutex , architecture independent
8. proc_a32.S: functions to enable icache and dcache, needs to be re-written.
9. ssvce_a32.S: ARM Secure services library, needs to be completely re written for a port.
10. static_ta.c : architecture independent
11. tee_l2cc_mutex.c: ARM TrustZone specific code, allows for shared mutex, I think implementing this should be optional.
12. tee_ta_manager.c: Manages linking and unlinking Trusted Applications, looks architecture independent for the most part.
13. tee_time.c: functions to get tee time and rich OS time. Should not be hard to implement.
14. tee_time_ree.c: Architecture independent.
15. thread.c:multithreading, architecture dependant, must change register handling when switching threads.
16. user_ta.c : Architecture independent
17. vfp.c : In accordance to ARM vector floating point, must change it to match with RISC-V
18. wait_queue: architecture independent.

Most of the kernel code needs to be ported over to RISC-V if we are going for a full port.
This covers most of the files, I have not included some files as they are extensions of other files or they are not important.

[6]https://github.com/OP-TEE/optee_client
[7]https://github.com/OP-TEE/optee_linuxdriver
[8]https://github.com/OP-TEE/optee_os
[9]https://github.com/OP-TEE/optee_test
[10]https://github.com/OP-TEE/build

core/arch/arm/include:
Header files.

core/arch/arm/mm (memory management):
Code for memory management, some of it can be reused, but there is some arm specific code.

core/arch/arm/sm:
Code related to the secure monitor.
This part is tricky, but in any case it depends on what implementation we are going to use, it basically handles context switching between normal and secure worlds.

core/arch/arm/sta:
Some tests for API and core

core/arch/arm/tee:
arch_svc.c and arch_svc_a32.S: syscall interface, lot of ARM based code.
Other files are also architecture dependent code.

core/lib/libtomcrypt:
- Architecture independent library.
- Library for cryptographic functions like one-way hash and encryption algorithms.

core/kernel:
Architecture independent, defines interrupts and other functions.

lib:
libmpa:
Basic arithmetic functions defined, part of it ARM specific, should be re-written to RISC-V.

libutee:
TEE library, most (80%) of it looks like it’s directly portable.

libutils:
Contains some ISO C functions like snprintf which are directly portable.

5. IMPLEMENTATION

5.1 Chromium

Google's Chrome is based on the open source Chromium project [15]. The Chromium browser's Pepper based CDM[11] (PPAPI) interfaces are open source as well.

5.1.1 Overview

Chromium could be analyzed into three main parts: the browser, the renderer and the WebKit. The browser accounts for the main process and also the user interfaces and I/O. The renderer plays the sub process role that is driven by the browser. And this embeds the Webkit to do the rendering process and layout. The browser and the renderer communicate through Chromium IPC system [16].

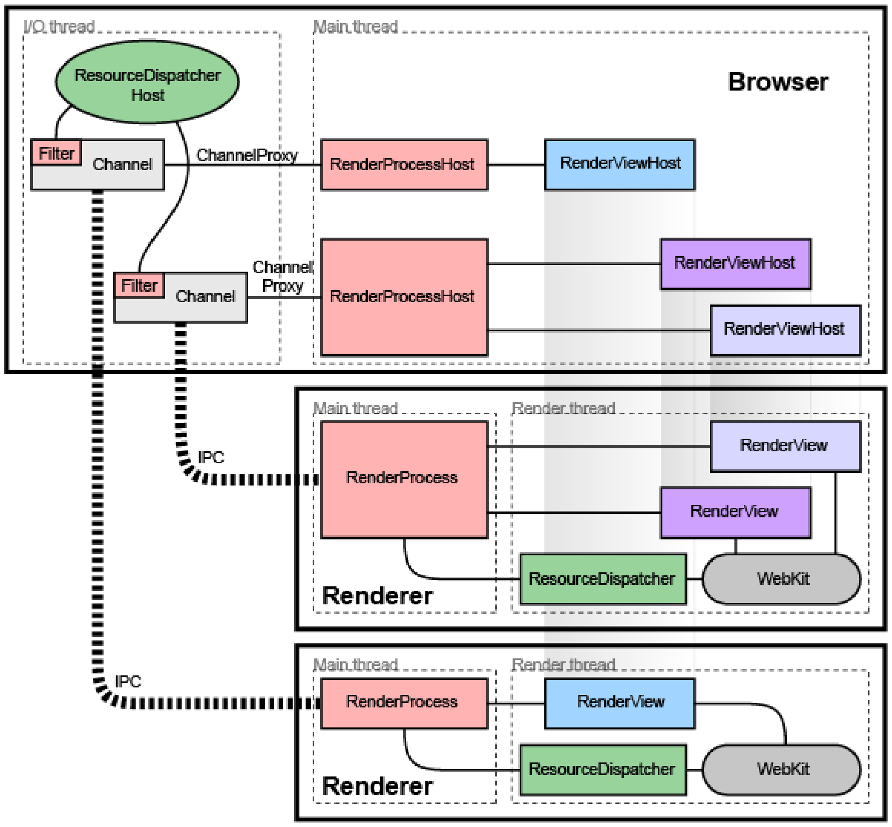


Figure 9: Chromium multi-process architecture [17]

[11]https://groups.google.com/a/chromium.org/d/msg/chromium-dev/exotX6Nf_z0/UtMi4a2sLncJ

Application layers:

The conceptual layer is as shown in the below figure-10 as to how the web pages are displayed in Chromium from the bottom to up.

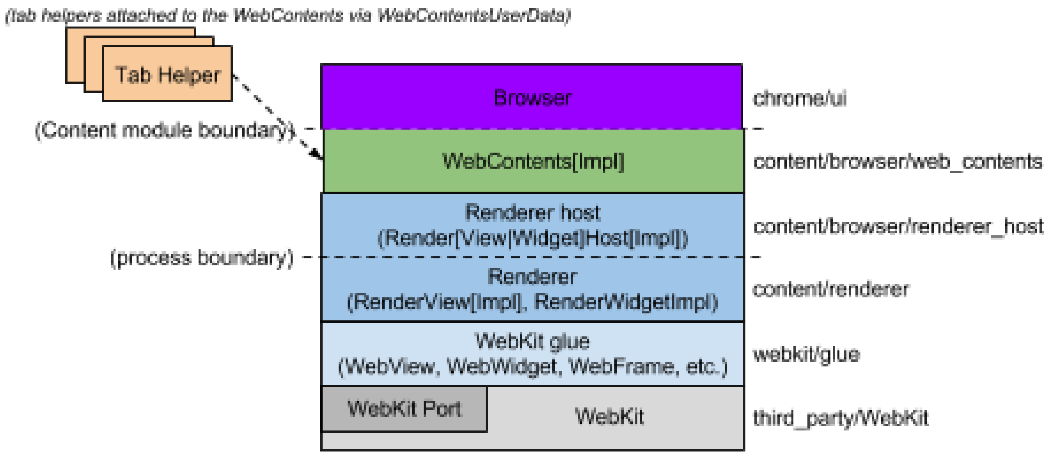

(tab helpers attached to the WebContents via WebContentsUserData)
Tab Helper
(Content module boundary)
(process boundary)
Browser
chrome/ui
WebContents[Impl]
content/browser/web_contents
Renderer host
(Render[View|Widget]Host[Impl])
content/browser/renderer_host
Renderer
(RenderView[Impl], RenderWidgetImpl)
content/renderer
WebKit glue
(WebView, WebWidget, WebFrame, etc.)
webkit/glue
WebKit Port
WebKit
third_party/WebKit

Figure 10: Web-page display process in chrome [18]

The WebKit:

The WebKit is basically a rendering engine. The WebKit “glue” layer provides an embedding API for WebKit that encloses the Chromium code base from WebCore data types to help minimize the impact of the changes in the Chromium code base.

The browser processes:

All the low level browser process objects communicate through IPC mechanism and are done on the I/O thread of the browser. When the *RenderProcessHost* is initialized on the main thread, a new object for the *Channelproxy* runs on the I/O thread. This object automatically forwards all the messages back to the *RenderProcessHost* on the UI thread at the main browser. The *ResourceMessageFilter* filters out the messages that can be handled by the I/O thread.

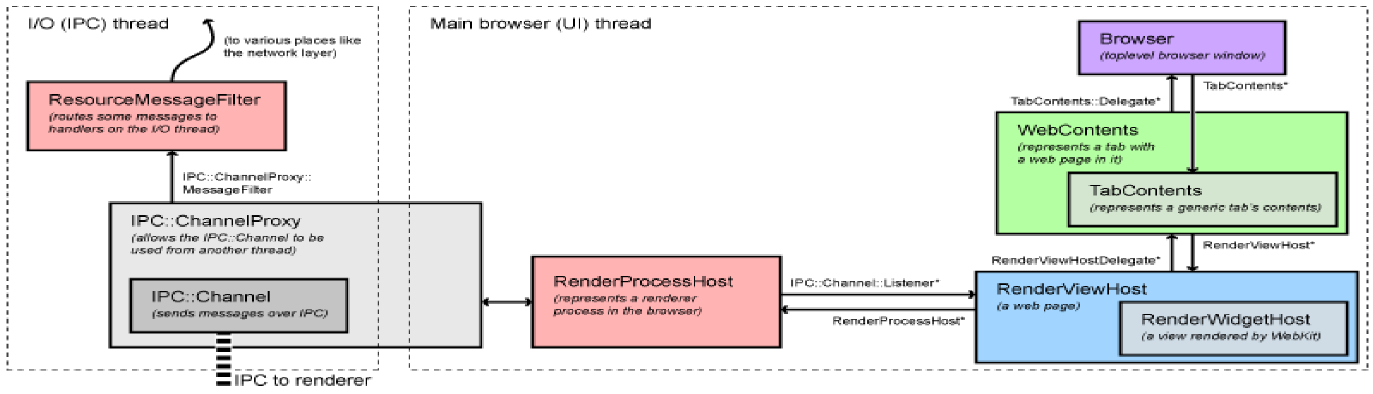

I/O (IPC) thread
(to various places like the network layer)
ResourceMessageFilter
(routes some messages to handlers on the I/O thread)
IPC::ChannelProxy::MessageFilter
IPC::ChannelProxy
(allows the IPC::Channel to be used from another thread)
IPC::Channel
(sends messages over IPC)
IPC to renderer
Main browser (UI) thread
Browser
(toplevel browser window)
TabContents*
TabContents::Delegate*
WebContents
(represents a tab with a web page in it)
TabContents
(represents a generic tab's contents)
RenderViewHost*
RenderViewHostDelegate*
RenderProcessHost
(represents a renderer process in the browser)
IPC::Channel::Listener*
RenderProcessHost*
RenderViewHost
(a web page)
RenderWidgetHost
(a view rendered by WebKit)

Figure 11: Chromium browser IPC thread [18]

The render process:

Chromium’s rendering process embeds the WebKit port using the “glue” interface. The *RenderView* object represents the web page that handles all navigation related commands. Each renderer has two threads which is required for communicating the messages to the browser process.

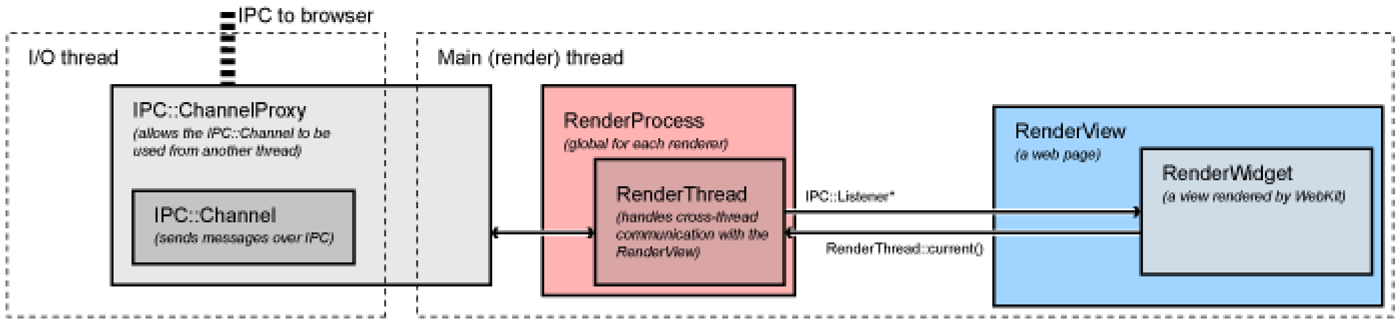

IPC to browser
I/O thread
Main (render) thread
IPC::ChannelProxy
(allows the IPC::Channel to be used from another thread)
IPC::Channel
(sends messages over IPC)
RenderProcess
(global for each renderer)
RenderThread
(handles cross-thread communication with the RenderView)
IPC::Listener*
RenderThread::current()
RenderView
(a web page)
RenderWidget
(a view rendered by WebKit)

Figure 12: Chromium Rendering process [18]

5.1.2 PPAPI and ffmpeg

PPAPI is a plug-in API developed as a successor of NPAPI to make it more secure and portable, but not limiting to the process of moving the plug-ins out [19][20]. The features of the PPAPI resemble the features of the JavaScript in HTML5 standard.

Life cycle of a plug-in[12] -> renderer call

1. Plug-in calls PPAPI function.
2. Thunk layer converts this to a C++ call on the proxy resource object.
3. Proxy resource does a *CallRenderer* with its message. This gets embedded into a "resource call" IPC message which encodes the resource ID and instance.
4. The *ResourceHost* in the renderer receives the message and finds the corresponding resource host object.
5. The *ResourceHost* decodes the message and performs the operation.

[12]https://www.chromium.org/developers/design-documents/pepper-plugin-implementation

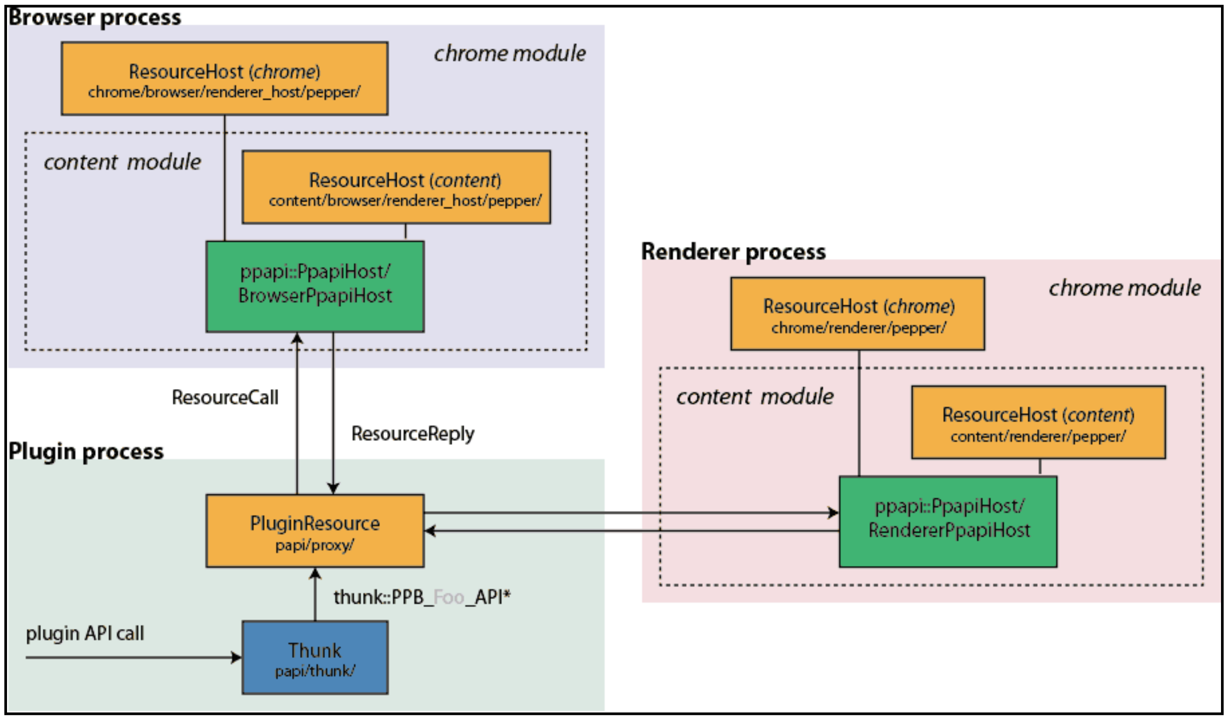


Figure 13: Lifecycle of plug-in in a browser

FFmpeg is a very fast video and audio converter that can convert arbitrary sample rates with the help of a high quality polyphase filter. The FFmpeg reads the input files and get the streaming packets containing the encoded data. These encoded packets are then passed to the decoder. The decoder produces the uncompressed raw frames of the samples which are processed by filtering. After filtering, the samples are passed to the encoder for encoding and the outputs the encoded streaming packets. These packets are passed to the muxer, which sends the samples to the output file. The overview of the FFmpeg process is as shown in the below figure-14:

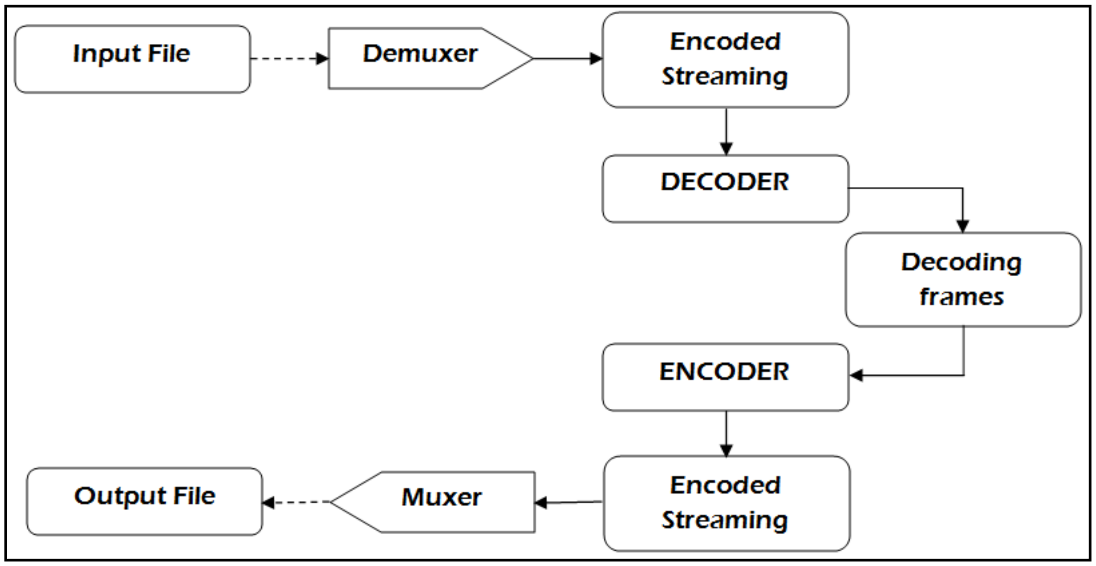


Figure 14: FFmpeg decoding process

5.1.3 OCDM Architecture

The Open Content decryption Module is a Content Decryption module developed as per the W3C EME specification [21]. The open source module is published on GitHub and maintained by Fraunhofer FOKUS under a (permissive) apache License, Version 2.0.

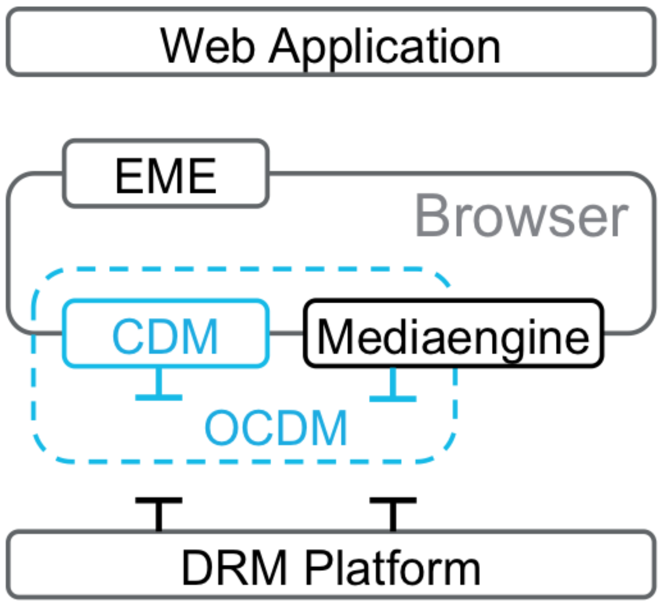


Figure 15: OCDM Architecture [22]

The CDM interoperability could take three forms- Common CDM, Common DRM and Common Encryption [23].

The OCDM requires an interface in the web browser and an EME implementation. This implementation of the interface enables the calls to JavaScript-based EME API to be forwarded to OCDM. This is termed as "glue code" [24]. Currently, the GitHub repository offers this mechanism for Chromium version 43[13] leveraging its PPAPI interface. However, this thesis implements this mechanism for the latest Chromium version 53[14] and is ready to be published.

The architecture of the OCDM is designed using three different abstraction layers- Browser glue, Core and Communication categorized into components namely CDM and Mediaengine as shown in figure-16.

[13] https://github.com/fraunhoferfokus/open-content-decryption-module/blob/2357_ppapi_chromium/COMPATIBLE
[14] https://github.com/RohanKrishnamurthy/open-content-decryption-module/tree/2774_ppapi_chromium

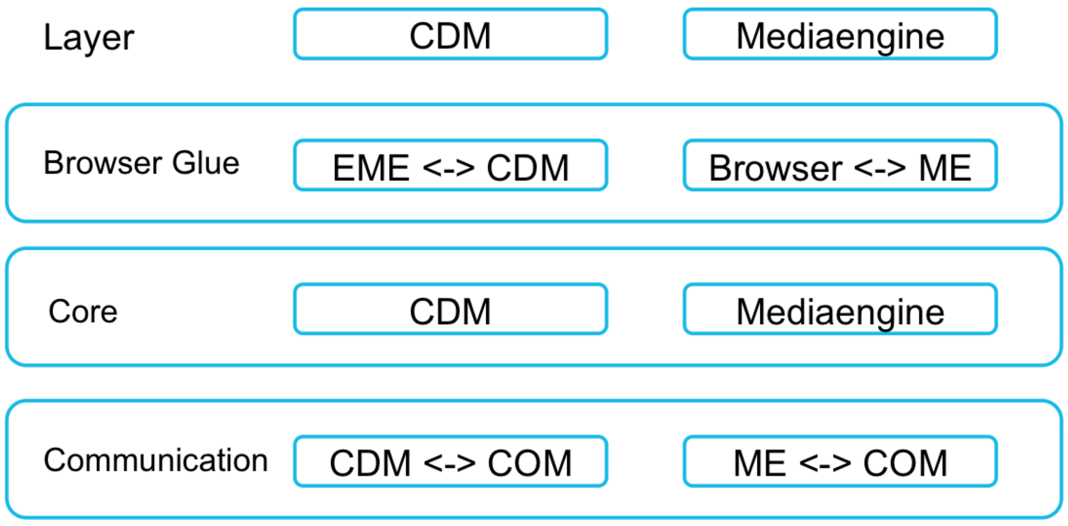


Figure 16: Layer OCDM Architecture [22]

The browser glue code depends on the CDM interface on the browser and is responsible for linking the CDM and Mediaengine. The core component implementations are located at src/cdm/[15] and src/mediaengine/[16] respectively. This implements the CDM interface and uses the OCDM core CDM component to establish the calls. These calls are mediated between the CDM and Mediaengine from the browser layer to the DRM system. The communication mechanisms are encapsulated by a communication interface and placed in src/com/[17] for CDM src/com/cdm/[18] and src/com/mediaengine/[19] respectively. The current OCDM implementation uses the Unix RPC mechanism to establish the communication and shared memory for IPC to a DRM system.

- **Note**: Implementations for other browser environments just need to implement the EME-based interfaces according to their environment and use the core components accordingly.

Within the repository the browser glue code can be found at src/browser/[20]. A PPAPI specific glue implementation for Chromium can be found in the OCDM repository at src/browser/chrome/[21]. This implements the Chrome-defined CDM interface and uses the OCDM core CDM component to forward the calls to a proper communication mechanism.

[15]https://github.com/fraunhoferfokus/open-content-decryption-module/tree/2357_ppapi_chromium/src/cdm
[16]https://github.com/fraunhoferfokus/open-content-decryption-module/tree/2357_ppapi_chromium/src/mediaengine
[17]https://github.com/fraunhoferfokus/open-content-decryption-module/tree/2357_ppapi_chromium/src/com
[18]https://github.com/fraunhoferfokus/open-content-decryption-module/tree/2357_ppapi_chromium/src/com/cdm
[19]https://github.com/fraunhoferfokus/open-content-decryption-module/tree/2357_ppapi_chromium/src/com/mediaengine
[20]https://github.com/fraunhoferfokus/open-content-decryption-module/tree/2357_ppapi_chromium/src/browser
[21]https://github.com/fraunhoferfokus/open-content-decryption-module/tree/2357_ppapi_chromium/src/browser/chrome

5.1.4 CDMi Specification

The current W3C EME specification describes an open interface to implement as EME-compliant Content Decryption Module (CDM) that provides an access to a platform DRM component that offers the support to the defined Content Decryption Module interface (CDMi).

The relationship of the CDMi interface with the browser, the media engine and the DRM system is shown in the below figure-17.

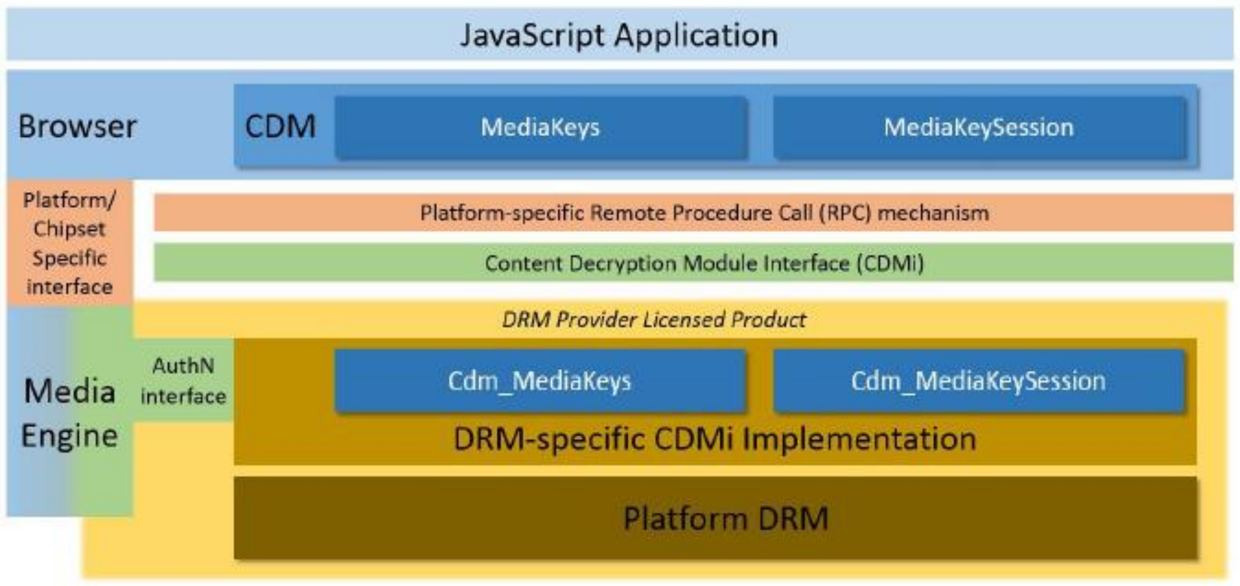

JavaScript Application
Browser
CDM
MediaKeys
MediaKeySession
Platform/ Chipset Specific interface
Platform-specific Remote Procedure Call (RPC) mechanism
Content Decryption Module Interface (CDMi)
DRM Provider Licensed Product
Media Engine
AuthN interface
Cdm_MediaKeys
Cdm_MediaKeySession
DRM-specific CDMi Implementation
Platform DRM

Figure 17: Content Decryption Module Interface Entity Relationship Diagram

The CDMi implementation is subject to compliant and robustness rules complying with the DRM provider's license agreement. This *DRM Provider License Product* contains the actual CDM which is used for Key acquisition.

The *Cdm_MediaKeys* and *Cdm_MediaKeySession* objects in the browser are the remote procedure calls (RPC) from the CDMi, using *Platform-specific Remote Procedure Call (RPC) mechanism.*

The *Media Engine* performs the decoding of the protected video streams outside of the browser. The media engine used as *Authenticated Interface* establishes a media Session to communicate with the CDMi implementation. This session ensures that only a trusted media engine may pass the media samples for decryption operation. The CDMi implementation enables the browser CDM complying as per the W3C EME standards without compromising content protection within the browser.

5.1.5 OCDMi

The OCDMi implementation is a CDMi-based DRM platform service which mocks the implementation to demonstrate the Microsoft PlayReady content playback in parallel to OCDM as shown in the below figure-18.

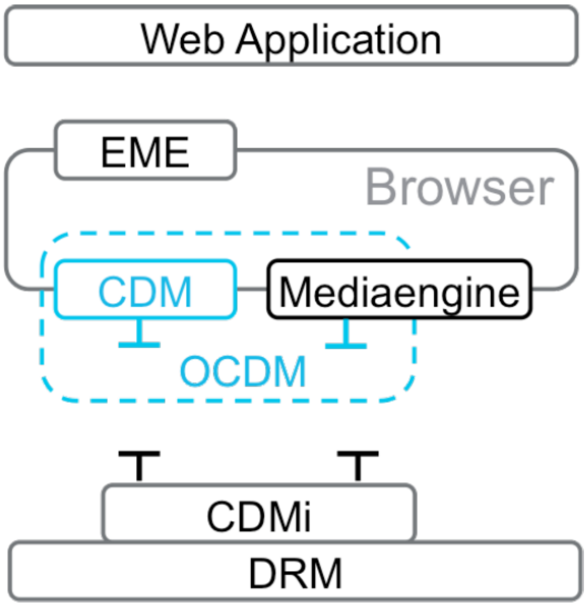


Figure 18: OCDMi architecture [22]

The OCDMi is also similar to the three-layer approach as supposed to OCDM but upside down. The communication layer mediate calls to the DRM specific glue code layer. The communication between OCDM and OCDMi is established using Remote Procedure Calls (RPC) mechanism. Any CDM that allows the interface to CDMi can be used along OCDMi for Microsoft PlayReady playback as shown in the figure-19.

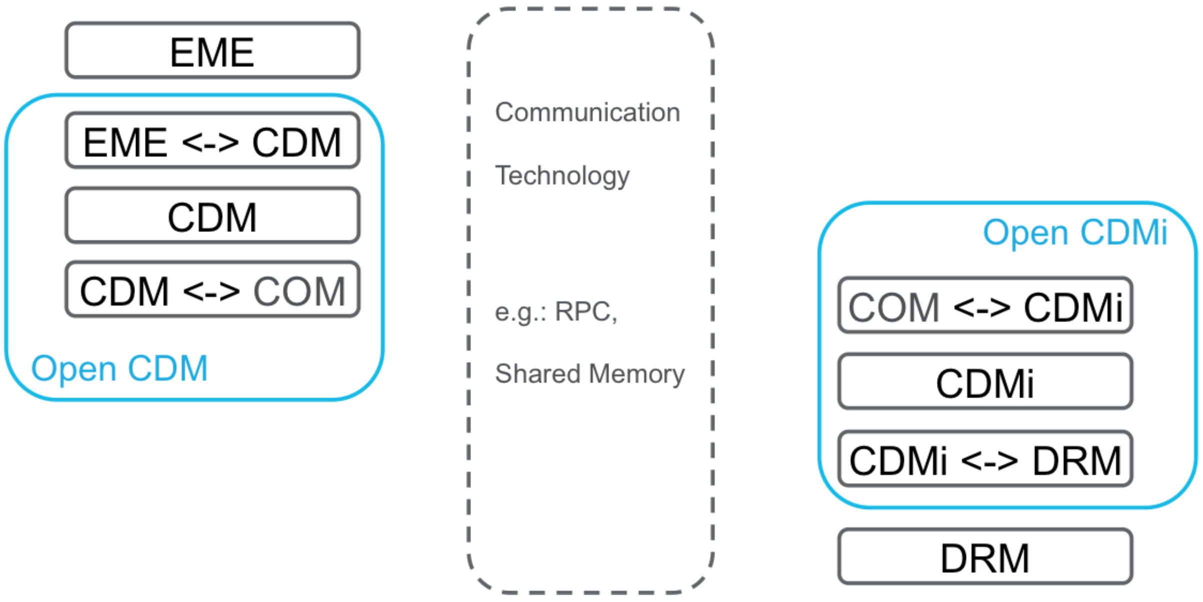
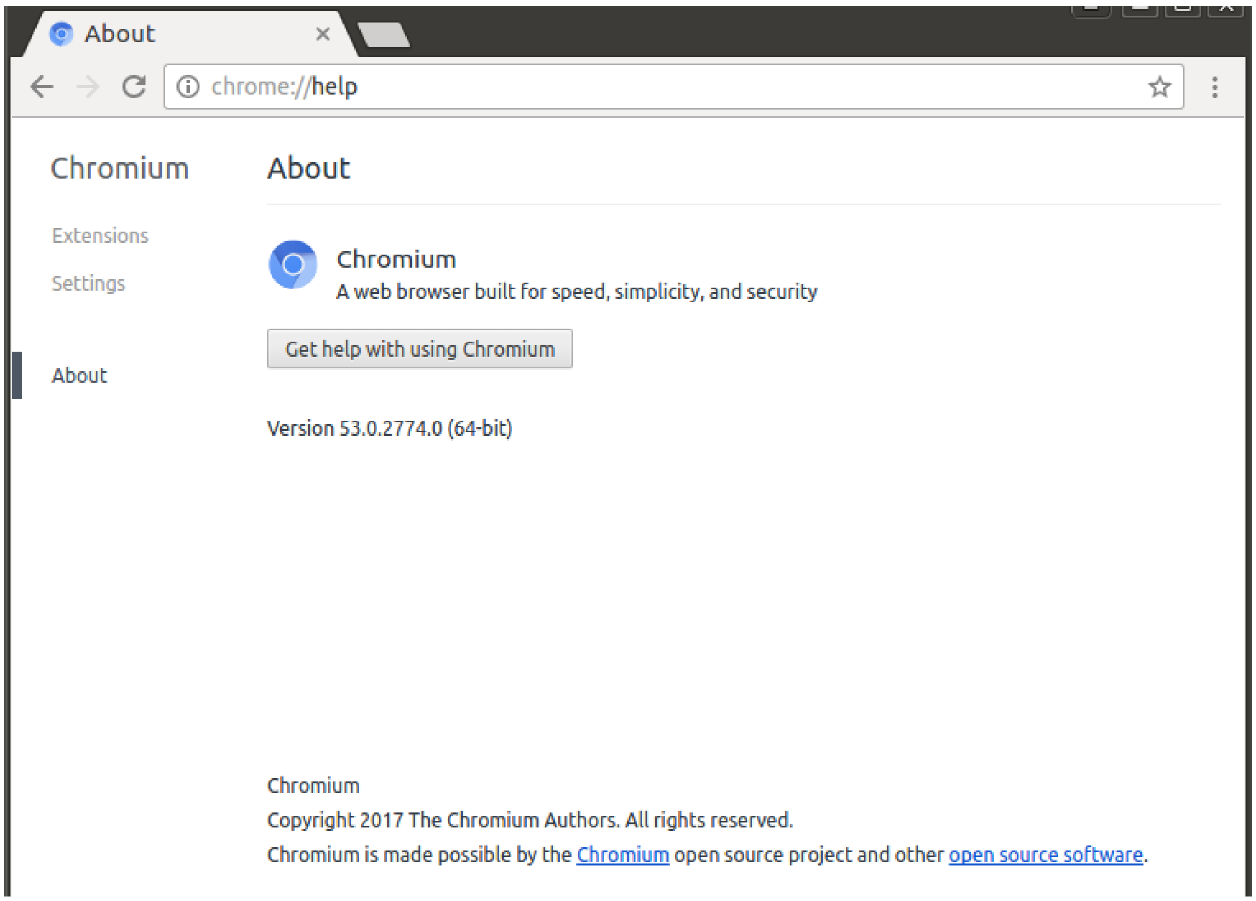


Figure 19: (CDM and CDMi) [22]

The OCDMi repository provides the implementation to demonstrate as to how a CDM can be adapted to be used with OCDM through the CDMi interface and enable the PlayReady playback. The implementation also involves the complete work flow between the Chromium browser itself and CDM.

5.1.6 Work Flow

- Initial Setup of Chromium
  - Set the Chromium Project work environment by the following the instructions: *https://chromium.googlesource.com/chromium/src/+/master/docs/linux_build_instructions.md*
  - Change the build to checkout the version 53.0.2774.0 using the command
    - *git checkout -b branch53 53.0.2774.0*
  - Run "*gclient sync --with_branch_heads --jobs 16*"to update the build with branch the current branch head.
  - Check the installation of Chromium by running the command
    - *out/Debug/chrome*



Chromium
A web browser built for speed, simplicity, and security

Version 53.0.2774.0 (64-bit)

Chromium
Copyright 2017 The Chromium Authors. All rights reserved.
Chromium is made possible by the Chromium open source project and other open source software.

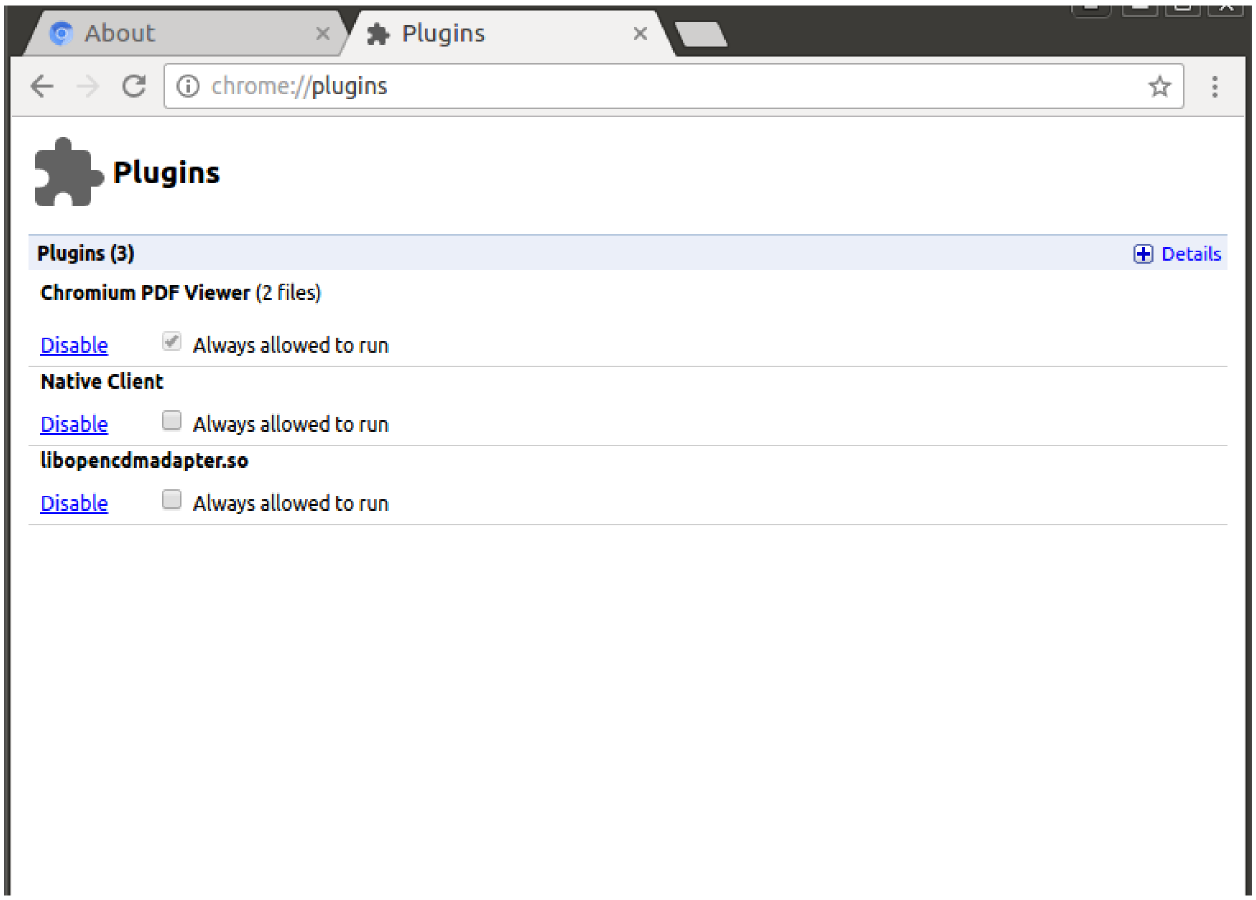
Figure 20: Snapshot of the Chromium version confirmation

➢ Setup of OCDM
- Follow the instruction from: Chromium –> open-content-decryption-module (https://github.com/RohanKrishnamurthy/Chromium) to download the OCDM repository
- Apply the patch to enable Microsoft PlayReady and provide the solution for the build version 53.

➢ Setup of OCDMi
- Follow the instruction from Chromium –> open-content-decryption-module-interface (https://github.com/RohanKrishnamurthy/Chromium) to download the OCDMi repository to run in parallel to OCDM.
- Run the Chromium:
    - o `./out/Debug/chrome --no-sandbox --register-pepper-plugins="out/Debug/libopencdmadapter.so;application/x-ppapi-open-cdm"`

Note: The Chromium browser must run with a "--no-sandbox" attribute and thus makes the playback insecure. The SANDBOX relies on Windows security manager to access the FAT32 files. And the RPC mechanism cannot instantiate the calls between the OCDM and OCDMi. Refer section 6.3.

- Check for the plug-in in the chrome://plugins and ensure that “libopencdmadapter” is enabled.



Plugins

Plugins (3) Details

Chromium PDF Viewer (2 files)

Disable ✓ Always allowed to run

Native Client

Disable ☐ Always allowed to run

libopencdmadapter.so

Disable ☐ Always allowed to run

Figure 21: Verification of addition of libopencdmadapter

➢ Microsoft PlayReady Porting Kit:
However, the implementation of the working protocol using the open sourced interface of the OCDM in conjunction with the PlayReady Porting Kit is closed source complying with the license formalities.
For more information on specifically PR integration please contact Fraunhofer FOKUS developer’s team: *http://www.fokus.fraunhofer.de/fame.*

➢ Snapshot

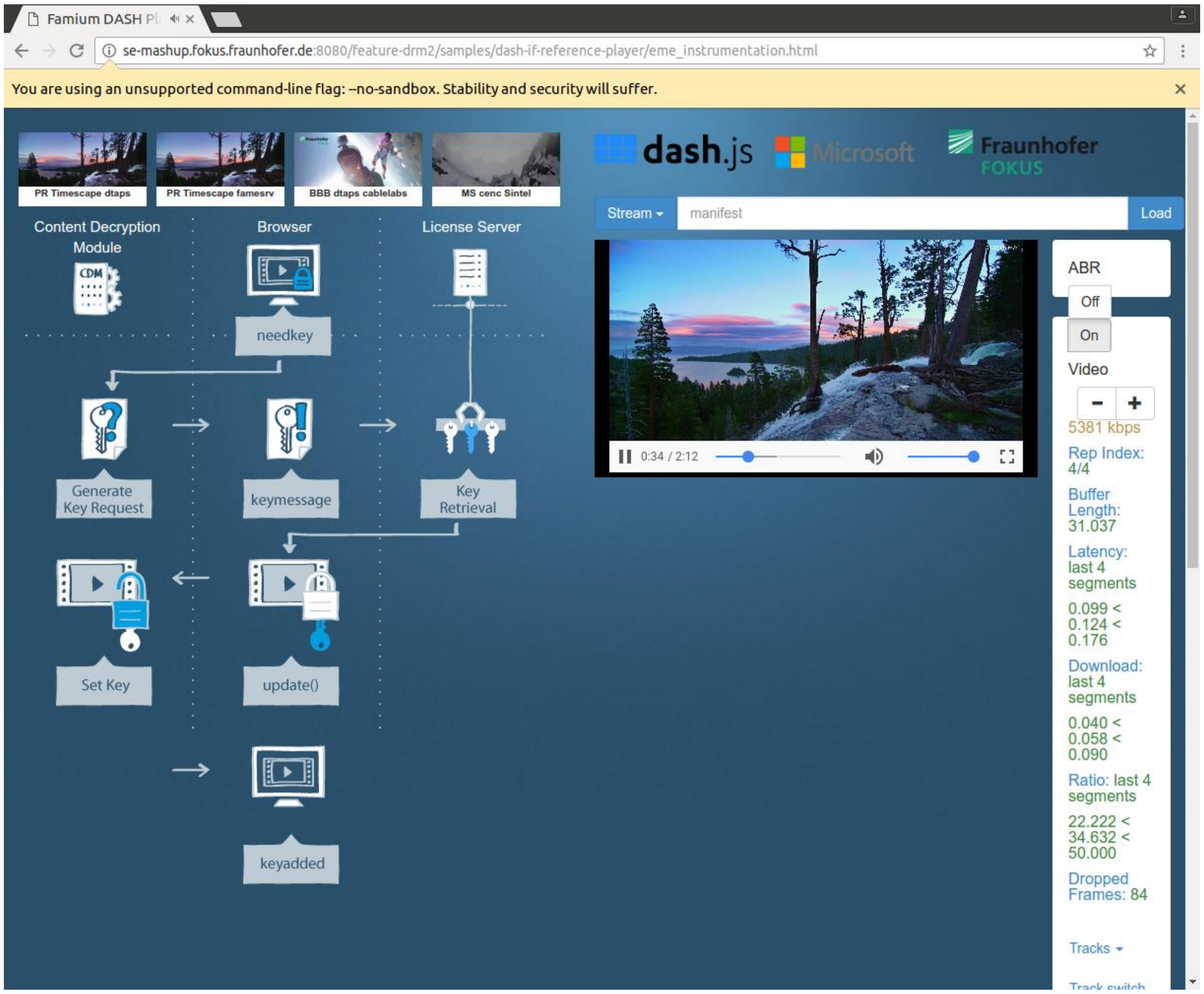


Figure 22: Playback of Microsoft PlayReady in Chromium

5.2 Firefox

5.2.1 Implementation of Microsoft Media Foundation based media player

The DRM based encrypted media file makes use of the protected media path (PMP) to communicate with the media foundation and acquires the DRM license automatically to enable the playback of the digital media. By default, the PMP process is operational within a Protected Environment (PE). This PE allows the isolation of the application process from the media pipeline. The phenomena of automatic rights acquisition refers to any action that the player window performs to paint the decrypted and decoded samples from the media foundation. The media foundation offers a generic mechanism termed as *IMFContentEnabler*[22]that performs this license acquisition.

However, the license acquisition is done outside of the PMP that is linked with player profile. The Media Session notifies the player through the *IMFContentProtectionManager*[23] interface to forward the content enabler object to the application. The *IMFContentEnabler* uses this interface and acquires the necessary rights.

The outline of the playback of the protected content using media foundations is as follows:

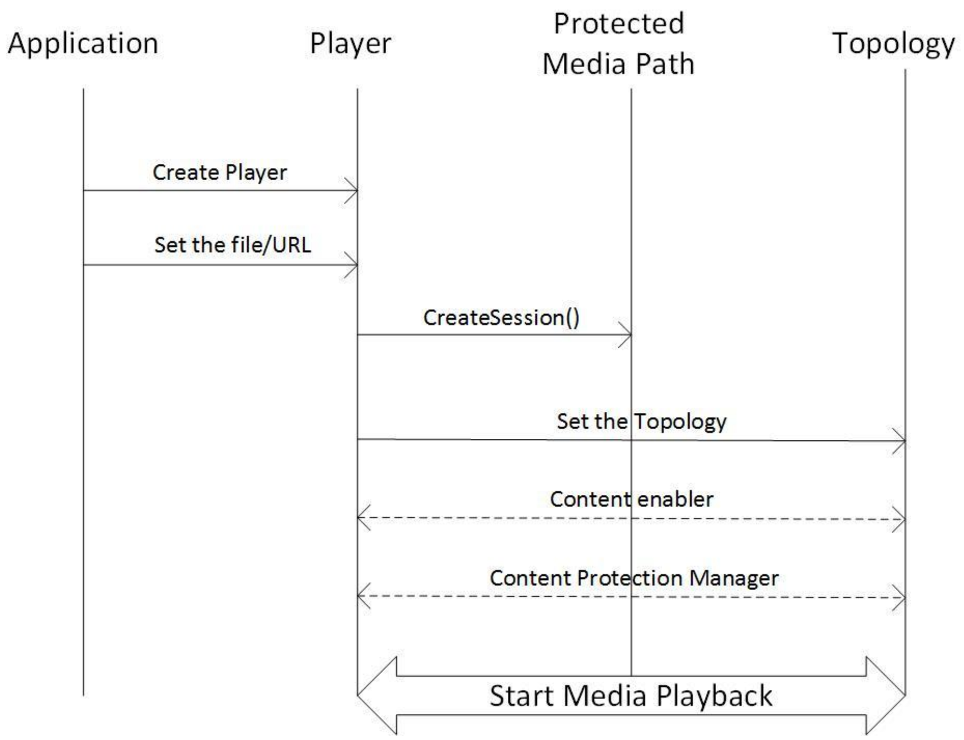


Figure 23: Playback of protected content using Media Foundation media player

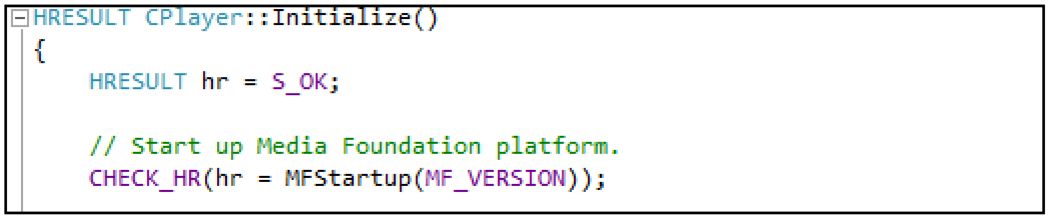
Algorithm of the call-flow:

[22]https://msdn.microsoft.com/en-us/library/windows/desktop/ms697004(v=vs.85).aspx
[23]https://msdn.microsoft.com/en-us/library/windows/desktop/ms694217(v=vs.85).aspx

1. The application sends a request for the creation of the media player and
2. The media player is fed with the file containing the protected video streaming samples
3. The media player creates a session using the method CreateSession()
4. The session is nothing but the secure media path known as Protected Media Path.
5. Set the topology with the input and output nodes for the secure pipeline
6. Call the ContentEnabler() and ContentProtectionManager() methods to acquire the DRM license
7. The media player is accessed with DRM rights to playback the protected video stream.

Code snippets and implementation of the algorithm is as follows:

➢ Initialize the Media Foundation
An instance is created to initialize the player object and this object initializes the media foundation by calling *MFStartup* function.



```
HRESULT CPlayer::Initialize()
{
    HRESULT hr = S_OK;

    // Start up Media Foundation platform.
    CHECK_HR(hr = MFStartup(MF_VERSION));
```

➢ Call the *MFCreatePMPMediaSession*
This creates a new instance of the Media Session.
   - The player receives a pointer to a proxy object which calls *IMFMediaSession.*
   - The proxy object then forwards all the calls to the media session.
   - Set the attributes (https://msdn.microsoft.com/en-us/library/windows/desktop/ms696989(v=vs.85).aspx) that is used to configure the objects, media formats and other purposes throughout the media foundation.
   - Set the *MF_SESSION_CONTENT_PROTECTION_MANAGER* attribute and store a pointer to the implementation of *IMFContentProtectionManager*.
   - Call *MFCreatePMPMediaSession* to create the media session.

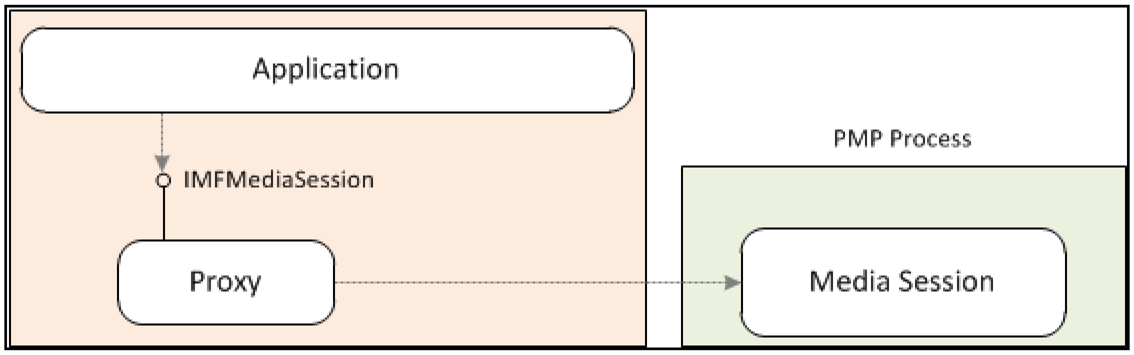
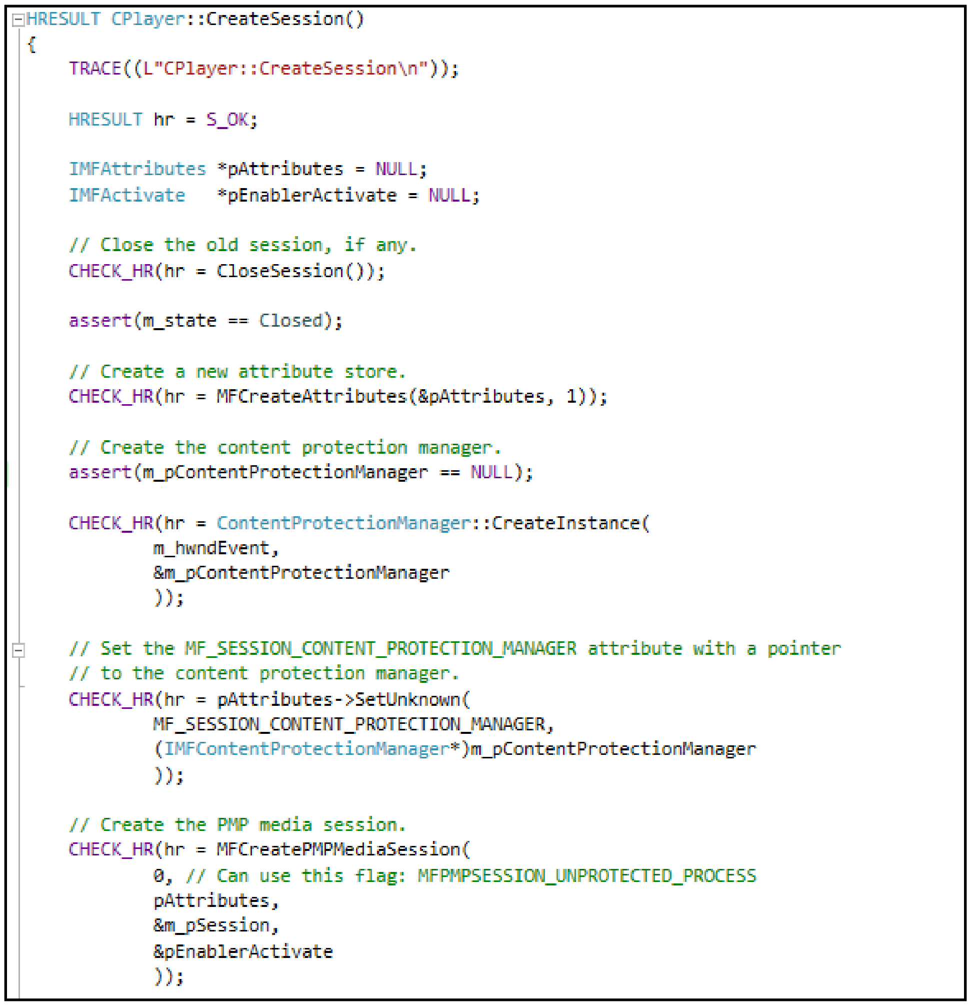


Figure 24: Call flow of PMPMediaSession [25]



```
HRESULT CPlayer::CreateSession()
{
    TRACE((L"CPlayer::CreateSession\n"));

    HRESULT hr = S_OK;

    IMFAttributes *pAttributes = NULL;
    IMFActivate   *pEnablerActivate = NULL;

    // Close the old session, if any.
    CHECK_HR(hr = CloseSession());

    assert(m_state == Closed);

    // Create a new attribute store.
    CHECK_HR(hr = MFCreateAttributes(&pAttributes, 1));

    // Create the content protection manager.
    assert(m_pContentProtectionManager == NULL);

    CHECK_HR(hr = ContentProtectionManager::CreateInstance(
            m_hwndEvent,
            &m_pContentProtectionManager
            ));

    // Set the MF_SESSION_CONTENT_PROTECTION_MANAGER attribute with a pointer
    // to the content protection manager.
    CHECK_HR(hr = pAttributes->SetUnknown(
            MF_SESSION_CONTENT_PROTECTION_MANAGER,
            (IMFContentProtectionManager*)m_pContentProtectionManager
            ));

    // Create the PMP media session.
    CHECK_HR(hr = MFCreatePMPMediaSession(
            0, // Can use this flag: MFPMPSESSION_UNPROTECTED_PROCESS
            pAttributes,
            &m_pSession,
            &pEnablerActivate
            ));
```

Figure 25: Snippet to create MF PMP Media Session

➢ Use the Source resolver

The source resolver takes the input as URL or a file data and creates the media source for the player. Call the *MFCreateSourceResolver*. The source resolver provides two methods: synchronous method, which has more responsive UI in comparison to asynchronous method which follows a different coding structure.

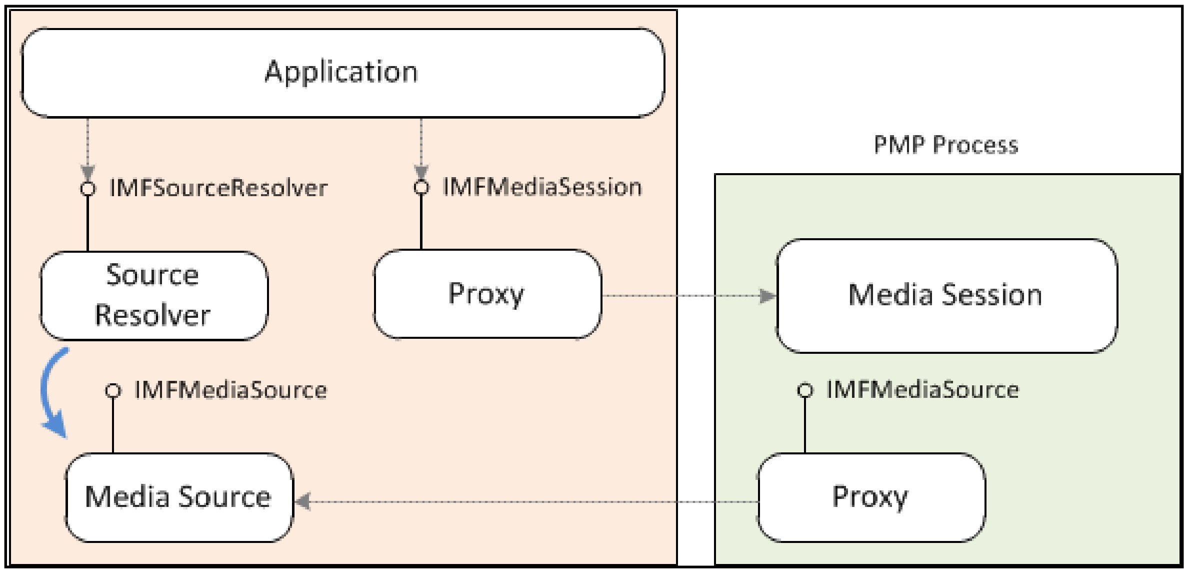
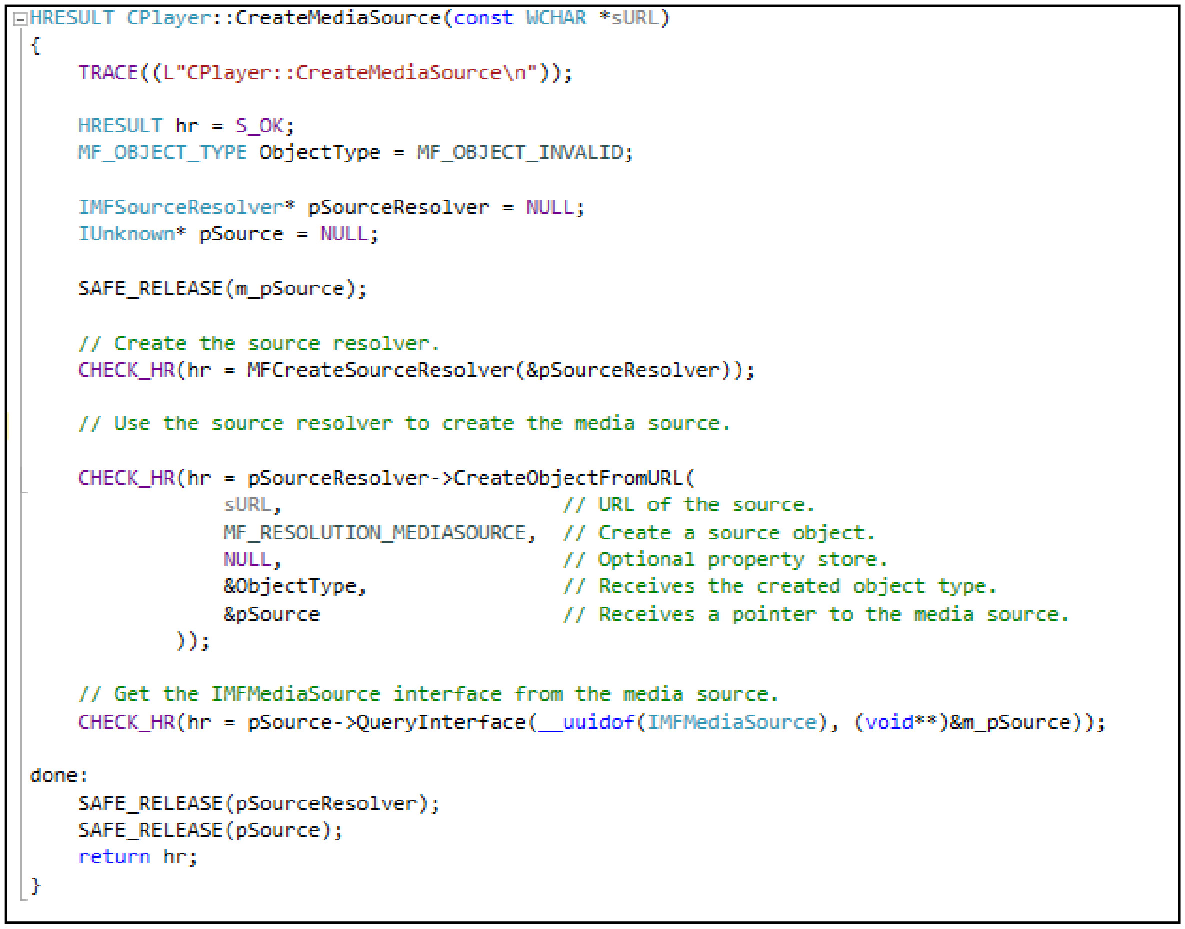


Figure 26: PMP Process [25]



```
HRESULT CPlayer::CreateMediaSource(const WCHAR *sURL)
{
    TRACE((L"CPlayer::CreateMediaSource\n"));

    HRESULT hr = S_OK;
    MF_OBJECT_TYPE ObjectType = MF_OBJECT_INVALID;

    IMFSourceResolver* pSourceResolver = NULL;
    IUnknown* pSource = NULL;

    SAFE_RELEASE(m_pSource);

    // Create the source resolver.
    CHECK_HR(hr = MFCreateSourceResolver(&pSourceResolver));

    // Use the source resolver to create the media source.

    CHECK_HR(hr = pSourceResolver->CreateObjectFromURL(
                    sURL,                       // URL of the source.
                    MF_RESOLUTION_MEDIASOURCE,  // Create a source object.
                    NULL,                       // Optional property store.
                    &ObjectType,                // Receives the created object type.
                    &pSource                    // Receives a pointer to the media source.
            ));

    // Get the IMFMediaSource interface from the media source.
    CHECK_HR(hr = pSource->QueryInterface(__uuidof(IMFMediaSource), (void**)&m_pSource));

done:
    SAFE_RELEASE(pSourceResolver);
    SAFE_RELEASE(pSource);
    return hr;
}
```

Figure 27: Snippet to create Media Source Resolver

- Create the Playback topology
  Once the source resolver is set to create the media source, get the source's presentation descriptor. To begin with,
  - Create an empty topology
  - For each stream descriptor, get the stream media type and create an object for the media sink, based on video or audio media type
  - Add a source node from the stream and an output node for the media sink and connect.

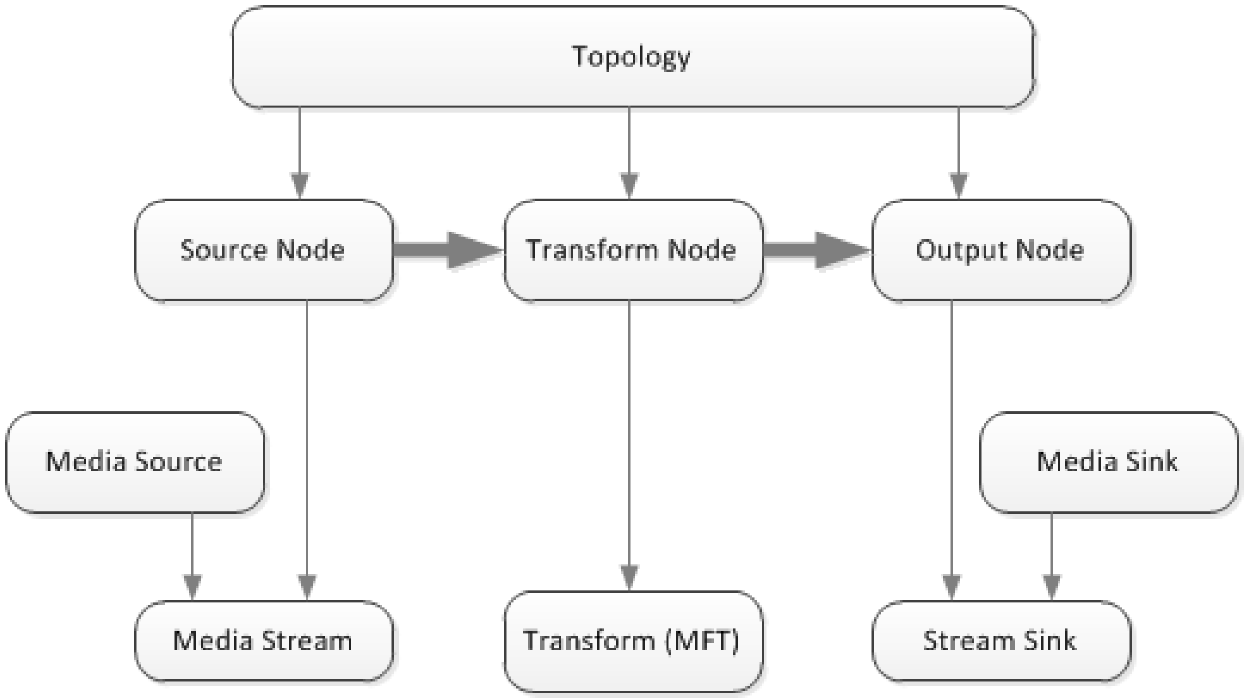
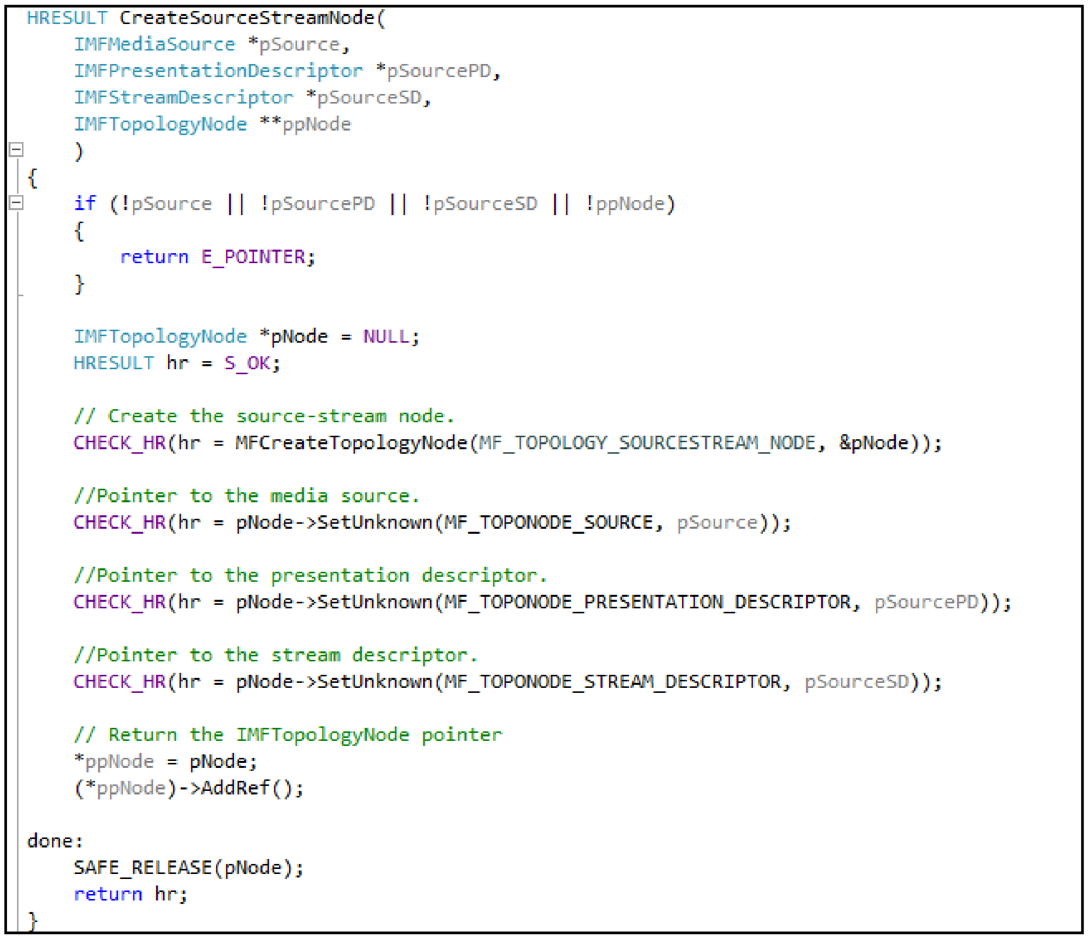


Figure 28: Media Foundation Topology [26]



```
HRESULT CreateSourceStreamNode(
    IMFMediaSource *pSource,
    IMFPresentationDescriptor *pSourcePD,
    IMFStreamDescriptor *pSourceSD,
    IMFTopologyNode **ppNode
    )
{
    if (!pSource || !pSourcePD || !pSourceSD || !ppNode)
    {
        return E_POINTER;
    }

    IMFTopologyNode *pNode = NULL;
    HRESULT hr = S_OK;

    // Create the source-stream node.
    CHECK_HR(hr = MFCreateTopologyNode(MF_TOPOLOGY_SOURCESTREAM_NODE, &pNode));

    //Pointer to the media source.
    CHECK_HR(hr = pNode->SetUnknown(MF_TOPONODE_SOURCE, pSource));

    //Pointer to the presentation descriptor.
    CHECK_HR(hr = pNode->SetUnknown(MF_TOPONODE_PRESENTATION_DESCRIPTOR, pSourcePD));

    //Pointer to the stream descriptor.
    CHECK_HR(hr = pNode->SetUnknown(MF_TOPONODE_STREAM_DESCRIPTOR, pSourceSD));

    // Return the IMFTopologyNode pointer
    *ppNode = pNode;
    (*ppNode)->AddRef();

done:
    SAFE_RELEASE(pNode);
    return hr;
}
```

Figure 29: Snippet to create MF Topology

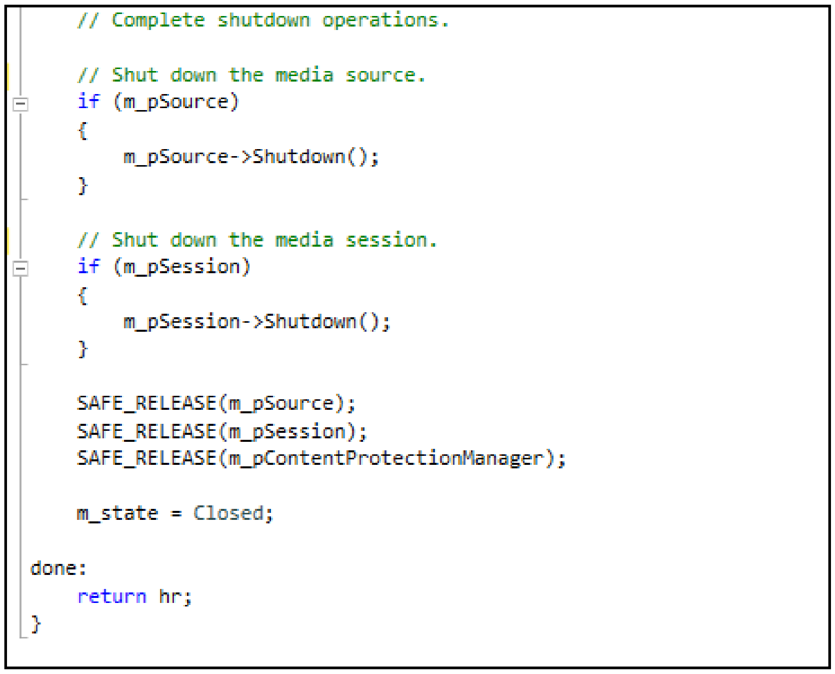
➢ Event generator and start playback
Use the *IMFMediaEventGenerator* interface to get the events from the media session and call the start the playback with play, pause and stop functions.

➢ Perform Rights Acquisition
- The Media Session calls the *IMFContentProtectionManager::BeginEnableContent* which redirects the pointer to *IMFActivate*interface.
- This interface creates the enabler object to perform the rights acquisition.
- The returned pointer from content enabler pointer uses the *IMFMediaEventGenerator*interface to get the events from the enabler object.
- Call the *IMFContentEnabler::AutomaticEnable* to perform license acquisition.
- When the function call is successful, the *MEEnablerCompleted* event completion notification is sent from content enabler.

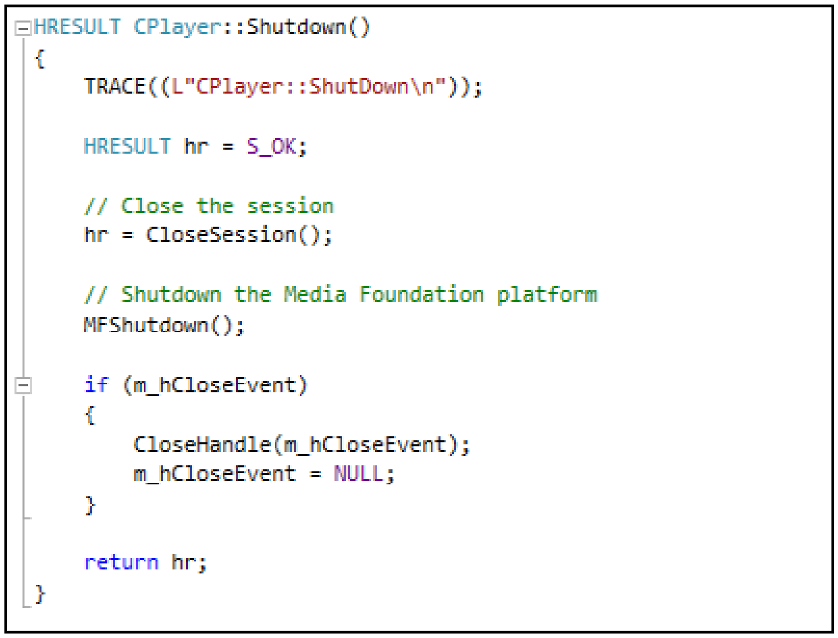
➢ Shutdown

When the application stops the playback,

- Shutdown the media session
- Shutdown the media source
- Shutdown the Media Foundation platform and close all the events.



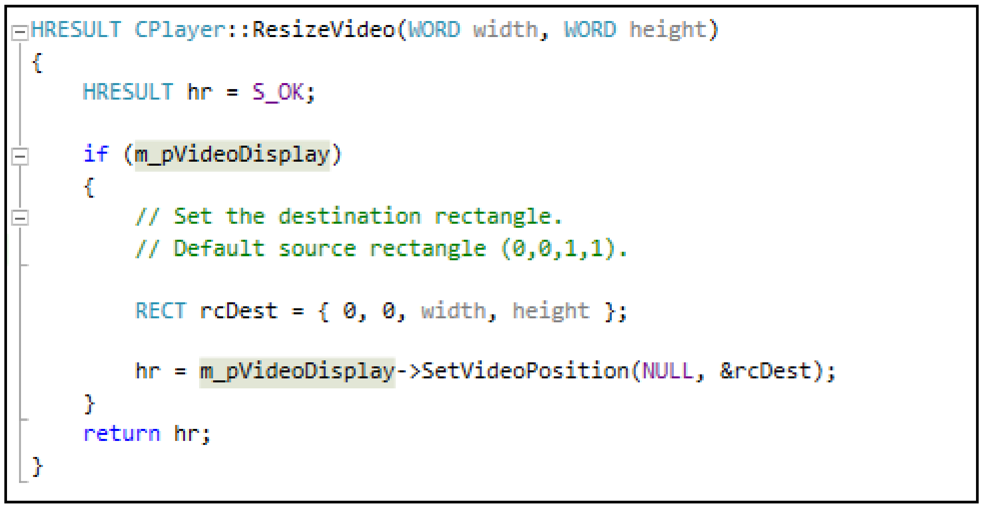
```
    // Complete shutdown operations.

    // Shut down the media source.
    if (m_pSource)
    {
        m_pSource->Shutdown();
    }

    // Shut down the media session.
    if (m_pSession)
    {
        m_pSession->Shutdown();
    }

    SAFE_RELEASE(m_pSource);
    SAFE_RELEASE(m_pSession);
    SAFE_RELEASE(m_pContentProtectionManager);

    m_state = Closed;

done:
    return hr;
}
```



```
HRESULT CPlayer::Shutdown()
{
    TRACE((L"CPlayer::ShutDown\n"));

    HRESULT hr = S_OK;

    // Close the session
    hr = CloseSession();

    // Shutdown the Media Foundation platform
    MFShutdown();

    if (m_hCloseEvent)
    {
        CloseHandle(m_hCloseEvent);
        m_hCloseEvent = NULL;
    }

    return hr;
}
```

Figure 30: Snippet to Shutdown() Media Foundation

- The Media Player
  - Set the video players height and width for display



```
HRESULT CPlayer::ResizeVideo(WORD width, WORD height)
{
    HRESULT hr = S_OK;

    if (m_pVideoDisplay)
    {
        // Set the destination rectangle.
        // Default source rectangle (0,0,1,1).

        RECT rcDest = { 0, 0, width, height };

        hr = m_pVideoDisplay->SetVideoPosition(NULL, &rcDest);
    }
    return hr;
}
```

Figure 31: Snippet to create the Media Foundation media player

- Snapshot

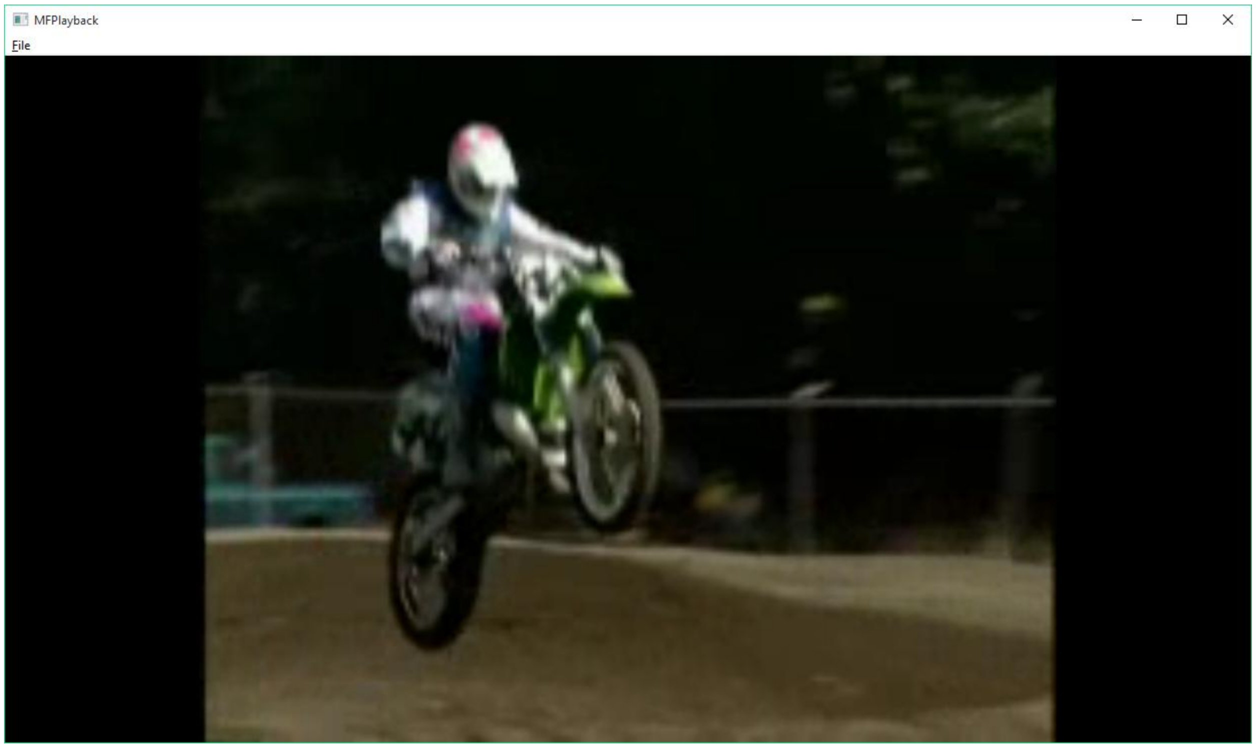


Figure 32: Snapshot of Playback of protected content using Media Foundation

5.2.2 Implementation of Mozilla Firefox – Nightly

The mapping of MSE and EME function calls from the browser's JavaScript API to the platform DRM component is exposed through Microsoft Media Foundation (MF). The CDM within the Firefox browser is responsible for managing the function calls to the Media Foundation and is integrated into the Firefox through the gecko Media Plug-in (GMP) interface as mentioned in sub chapter so and so. The GMP's are loaded as DLL files onto the browser on startup. For security reasons, the browser allows only a set of Media Foundation dll's as whitelist that allows the access to a fixed set of API's. For example, the Clearkey GMP CDM implementation[24]uses Microsoft Media Foundation to support h.264 video codec and AAC audio codec for decoding.

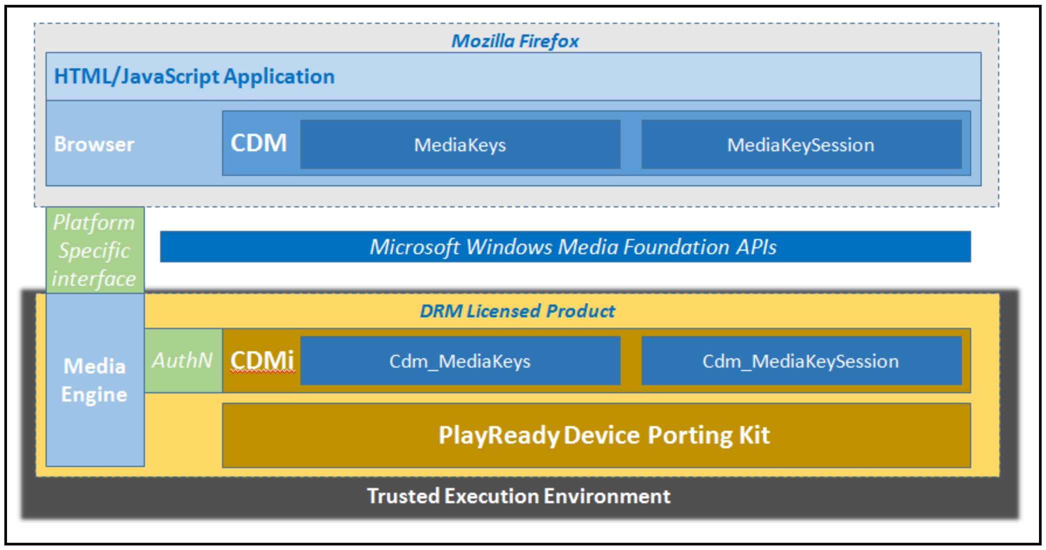

Mozilla Firefox
HTML/JavaScript Application
Browser
CDM
MediaKeys
MediaKeySession
Platform Specific interface
Microsoft Windows Media Foundation APIs
DRM Licensed Product
Media Engine
AuthN
CDMi
Cdm_MediaKeys
Cdm_MediaKeySession
PlayReady Device Porting Kit
Trusted Execution Environment

Figure 33: Enabling Open Source operability with Platform DRM

The Firefox browser implements MSE and EME API'S onto the Media Foundation to support Microsoft PlayReady EME. The mapping of the EME implementations includes the 4 main external components:

- Key System: The first step of mapping the EME is to identify the KeySystem. Apart from the default ClearKey, the EME must identify the PlayReady KeySystem.
- Content decryption Module (CDM): Enables the playback of encrypted media and provides an interface for the applications.
- License Server: The server communicates with the CDM and provides the keys to decrypt media samples.
- Packaging: Encodes and encrypts the media samples for distribution.

[24]https://github.com/cpearce/gmp-clearkey

For example, Netflix must authenticate its customers within their browser window; when the customer accesses the application, the application identifies the customer's identity and access rights.

The mapping of media foundation API onto EME is as shown below.

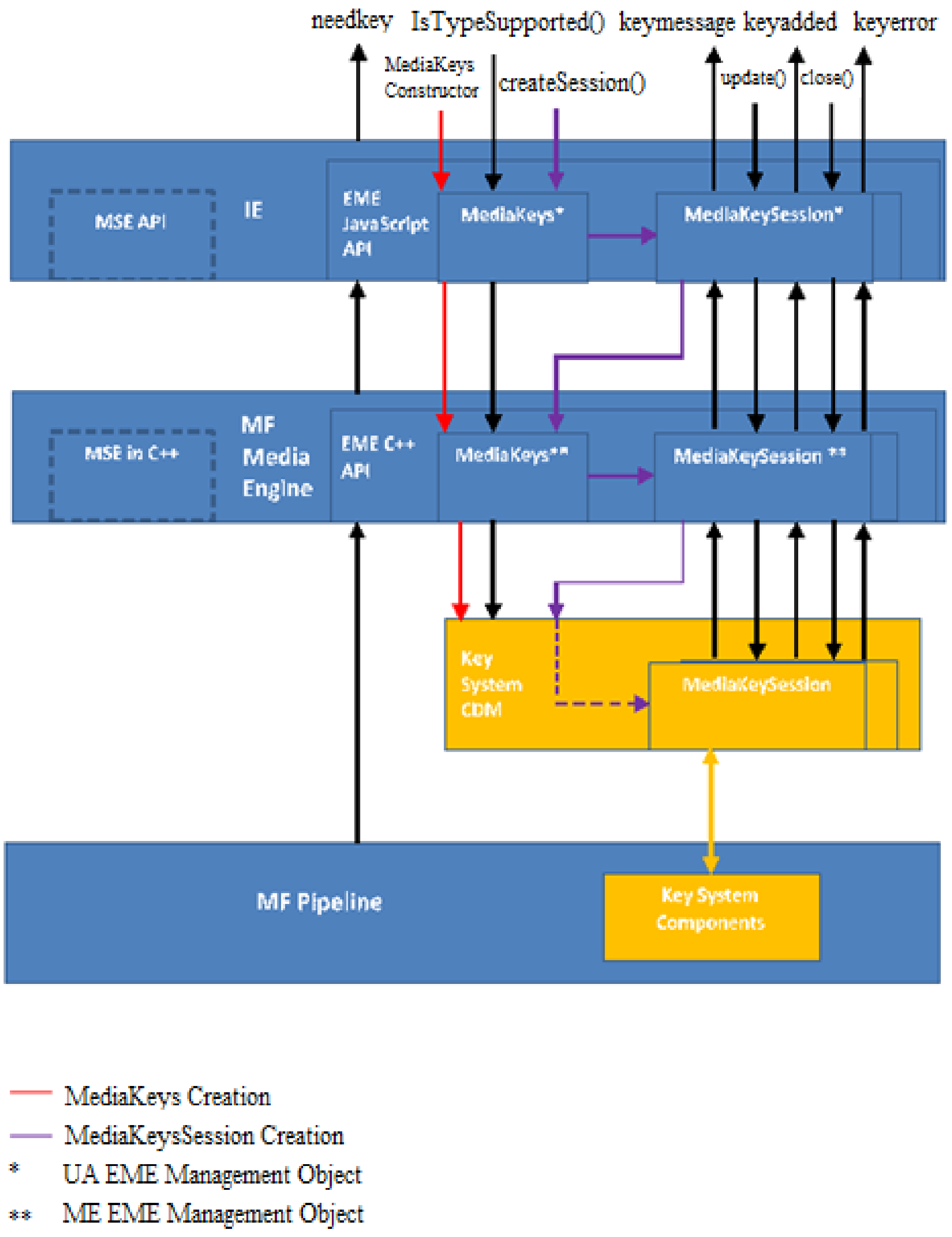


Figure 34: EME Call flow with Media Foundation

The call flow:

1. The application is loaded with encrypted streams.
2. The browser upon identifying the encrypted stream, calls for an encrypted event with metadata (obtained from the media).
3. During the encrypted event, check for the available key systems using the navigator.requestMediaKeySystemAccess()[25] and create an object using MediaKeySystemAccess. The MediaKeys object represents the keys available to decrypt the media. This object is nothing but a CDM instance specifically to have the interface with the creation of media key session, which is later used to obtain the license keys from the DRM server upon the creation of the MediaKeys object, the media element is associated with setMediaKeys()
4. The MediaKey Session is created by calling CreateSession() which also represents the usage policies and rights of the license and its key.
5. The application then generates a license request by calling generaterequest() on the MediaKeySession.
6. The CDM then sends a message request to acquire key from the license server.
7. The MediaKeySession object upon receiving the request, sends the message to the server.
8. The application receives the license from the server and sends the data to CDM using update() method.
9. The CDM accesses the key and decrypts the data.
10. Playback resumes

The appendix A and B[26] shows the APIs Mapping from MSE and EME in JavaScript to Media Foundation MediaEngine in C++.

Installing the build prerequisites

1. Install VS Community 2015[27] and also install the Common Tools for Visual C++ 2015, which requires a customized installation in VS 2015.
   Note: The Firefox codebase relies on C++ features in VS 2012 or later.

2. Download and install the latest MozillaBuild package[28] containing additional build tools. By default, the package installs to c:\mozilla-build. It is recommended to use the default path.

[25]https://w3c.github.io/encrypted-media/#navigator-extension-requestmediakeysystemaccess
[26]https://msdn.microsoft.com/en-us/library/windows/apps/dn466732.aspx
[27]https://www.visualstudio.com/downloads/#d-community

Get the Source code

1. After the prerequisites are installed, launch one of the batch files under the installed MozillaBuild (c:\mozilla-build by default):
   - start-shell-msvc2015.bat (recommended)

This will launch BASH command prompt and all further commands shall be executed in this shell.

2. Create a working for the source,
   - cd c:/
   - mkdir mozilla
   - cd mozilla/
3. Clone the repository
   - hg clone https://hg.mozilla.org/mozilla-central
4. Start the build process
   - ./mach build
5. Run the build
   - ./mach run
6. Enable Visual Studio for debugging
   - ./mach build-backend -b VisualStudio

     Note: The solution file can be found under the directory:

   - .\obj-i686-pc-mingw32\msvc\mozilla.sln

Gecko Media Plug-in

GMP is a special purpose extension point for authorized 3rd party codecs and EME (Encrypted Media Extensions) CDMs (Content Decryption Modules)[27]. Originally named as NGLayout, Gecko is the new term named after the layout engine which is under continuous development by Mozilla Project[29].

Gecko is an open source browser engine that supports open Internet standards such as HTML5, CSS, W3C, JavaScript and others. Gecko provides the foundation needed to display content on the screen, including a layout engine and a complementary set of browser components. Gecko is architectured from the ground up to the cross-platform. Licensing Gecko is royalty-free.

[28]https://ftp.mozilla.org/pub/mozilla.org/mozilla/libraries/win32/MozillaBuildSetup-latest.exehttps://ftp.mozilla.org/pub/mozilla.org/mozilla/libraries/win32/MozillaBuildSetup-Latest.exe
[29] www.mozilla.org

Base API

The API described in the code is located at: /dom/media/gmp/gmp-api in the browser's source code.

Following are the specific rules for naming conventions to load the GMP:

- The Plug-in directory name must start with "gmp-" and the rest of the name is called as the plug-ins "base-name". For ex. "gmp-playready" as used in this thesis.
- The dynamic library's name must be the plug-ins base name, for example: "playready.dll".
- The plug-in meta-data file is a UTF-8 text file with an ".info" after the base-name. For ex: "playready.info".

Build the dynamic library for PlayReady (playready.dll)

The DLL file exposes three main functionalities publicly:

- GMPInit
- GMPgetAPI
- GMPShutdown

The function types are as follows:

- typedefGMPErr (*GMPInitFunc)(void);
- typedef GMPErr (*GMPGetAPIFunc)(const char* apiName,void* aHostAPI, void** aPluginAPI);
- typedef void (*GMPShutdownFunc)(void);

Create the meta-data file

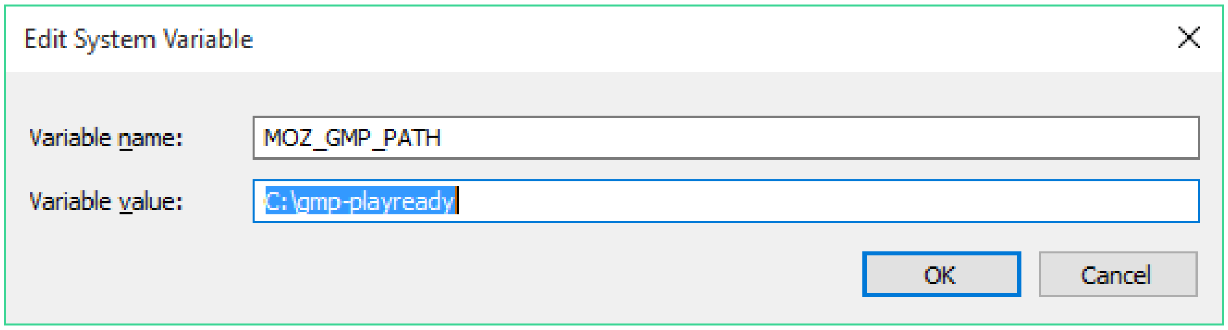
The GMP meta data is stored in a UTF-8 format wherein each line is a record, consisting of Name, Description, Version, and Records. As an example, below is the meta data file that is used to load GMP for PlayReady keysystem.

Load the Gecko Media Plug-in

As subjected in section so and so, the GMP's are special purpose extensions for the authorized 3rd party codecs and EME CDM's. The GMP essentially consists of a directory containing a dynamic library and a meta-data file.

GMP structure

- Set the system environment variable MOZ_GMP_PATH to specify the location of the playready.dll and the playready.info (meta) file, i.e.MOZ_GMP_PATH=/home/username/mozilla-central/media/gmp-playready
- This way, a list of known GMP's for different KeySystem can be distinguished and whenever there is a request for a different Key System from the application.



Variable name: MOZ_GMP_PATH
Variable value: C:\gmp-playready

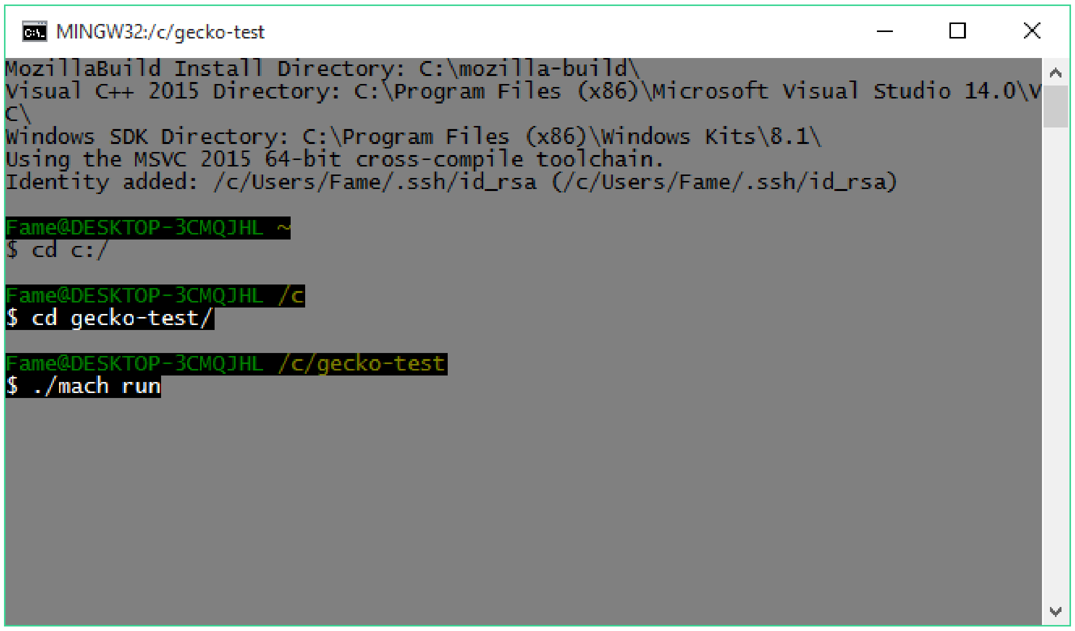
Figure 35: Load the GMP

Gecko then initiates a child process for the GMP and the child processes may be sandboxed. These child processes will load all the GMP's DLL into the main thread. Then as mentioned in the above section of "Build the dynamic library for PlayReady (playready.dll)", the three functions that are publicly exposed are invoked.

- GMPInit: initiation of GMP (with respect to playready.dll)
- GetGMPAPI: request for the desired base API
- GetShutdown: Closes or shuts down the child process from the main thread.

Launch the browser

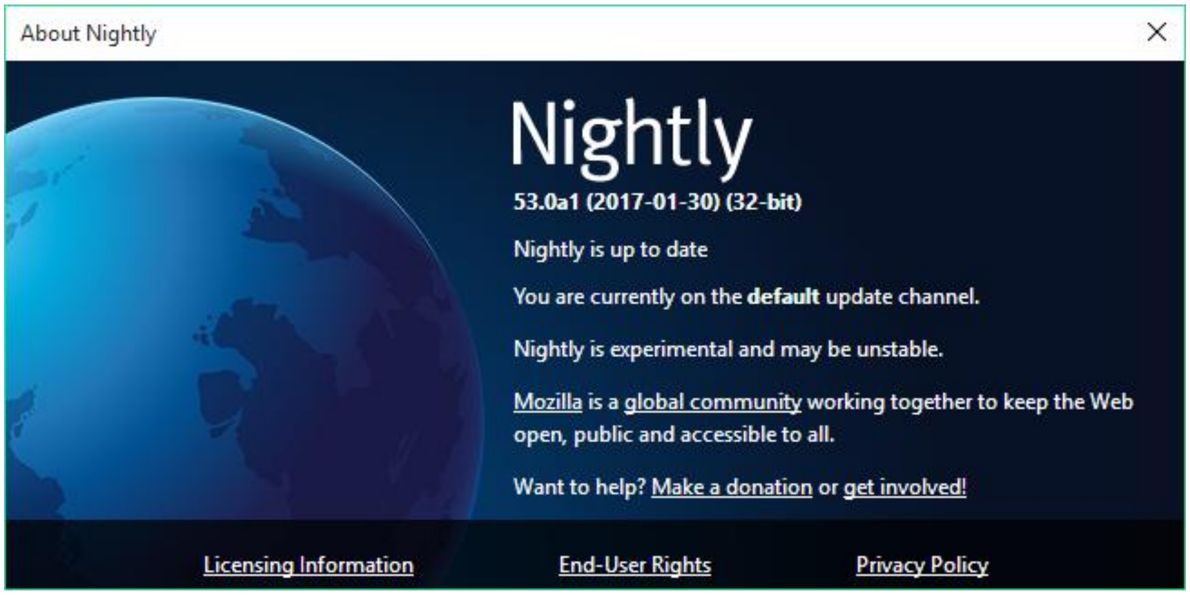
Run: start-shell-msvc2015.bat



```
MozillaBuild Install Directory: C:\mozilla-build\
Visual C++ 2015 Directory: C:\Program Files (x86)\Microsoft Visual Studio 14.0\V
C\
Windows SDK Directory: C:\Program Files (x86)\Windows Kits\8.1\
Using the MSVC 2015 64-bit cross-compile toolchain.
Identity added: /c/Users/Fame/.ssh/id_rsa (/c/Users/Fame/.ssh/id_rsa)

Fame@DESKTOP-3CMQJHL ~
$ cd c:/

Fame@DESKTOP-3CMQJHL /c
$ cd gecko-test/

Fame@DESKTOP-3CMQJHL /c/gecko-test
$ ./mach run
```

Figure 36: Launch the browser

The version of Nightly being implemented on is 53.0a1



About Nightly

Nightly

53.0a1 (2017-01-30) (32-bit)

Nightly is up to date

You are currently on the **default** update channel.

Nightly is experimental and may be unstable.

Mozilla is a global community working together to keep the Web open, public and accessible to all.

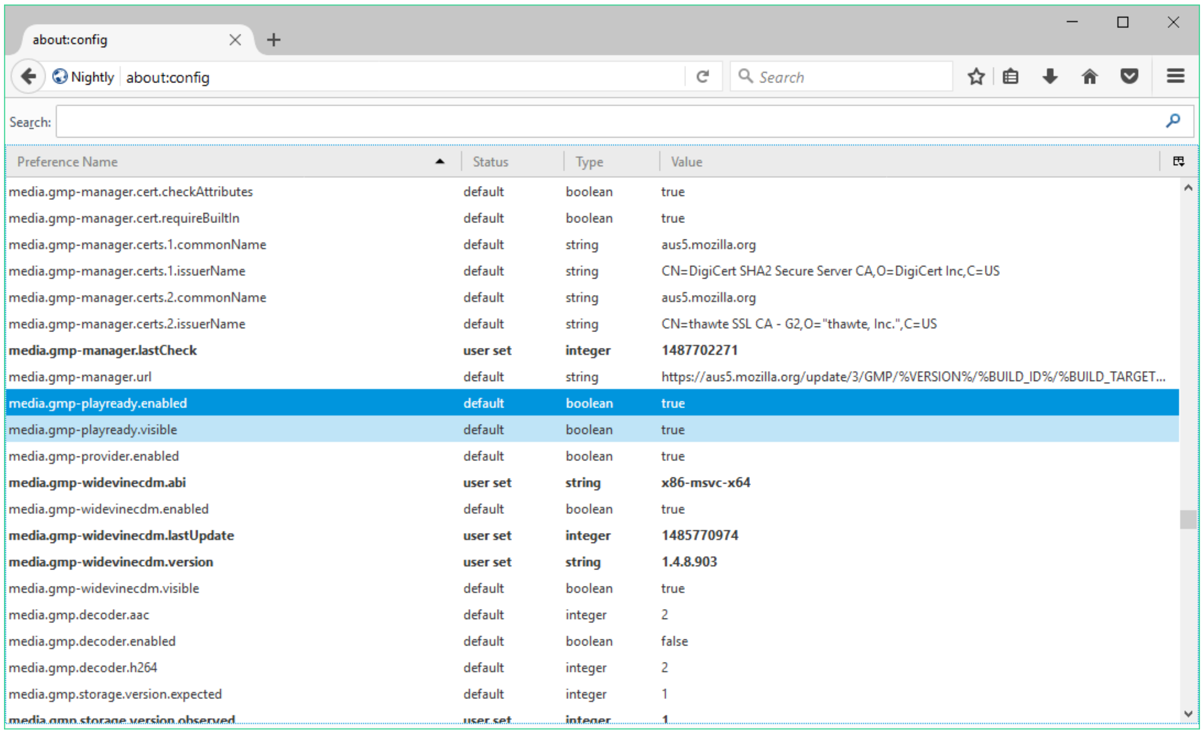
Want to help? Make a donation or get involved!

Figure 37: Nightly -version 53.0a1

The whole mozilla-central code base is also present at the online dxr repository maintained by Mozilla[30]:

Implementation plan involves the following processes:

1. Addition of PlayReady KeySystem
   - Involves patching of multiple lines of code in different file systems
   - The patched files are placed in the Github repository.
2. Create the plugin-id at the browser to confirm the availability and identification of the KeySystem
   The Firefox browser identified the PlayReady KeySystem and the also the addition of the plug-in support from the *about:config* URL is as shown in the figure-38.



Preference Name	Status	Type	Value
media.gmp-manager.cert.checkAttributes	default	boolean	true
media.gmp-manager.cert.requireBuiltIn	default	boolean	true
media.gmp-manager.certs.1.commonName	default	string	aus5.mozilla.org
media.gmp-manager.certs.1.issuerName	default	string	CN=DigiCert SHA2 Secure Server CA,O=DigiCert Inc,C=US
media.gmp-manager.certs.2.commonName	default	string	aus5.mozilla.org
media.gmp-manager.certs.2.issuerName	default	string	CN=thawte SSL CA - G2,O="thawte, Inc.",C=US
media.gmp-manager.lastCheck	**user set**	**integer**	**1487702271**
media.gmp-manager.url	default	string	https://aus5.mozilla.org/update/3/GMP/%VERSION%/%BUILD_ID%/%BUILD_TARGET...
media.gmp-playready.enabled	default	boolean	true
media.gmp-playready.visible	default	boolean	true
media.gmp-provider.enabled	default	boolean	true
media.gmp-widevinecdm.abi	**user set**	**string**	**x86-msvc-x64**
media.gmp-widevinecdm.enabled	default	boolean	true
media.gmp-widevinecdm.lastUpdate	**user set**	**integer**	**1485770974**
media.gmp-widevinecdm.version	**user set**	**string**	**1.4.8.903**
media.gmp-widevinecdm.visible	default	boolean	true
media.gmp.decoder.aac	default	integer	2
media.gmp.decoder.enabled	default	boolean	false
media.gmp.decoder.h264	default	integer	2
media.gmp.storage.version.expected	default	integer	1
media.gmp.storage.version.observed	**user set**	**integer**	**1**

Figure 38: Addition of PlayReady plug-in

3. Create the DLL: playready.dll
   - The project involving the creation .dll files is placed at the branch named "PlayReadyDLL" of the Github repository.
4. Create the .info meta file: playready.info
5. Load the GMP path with *playready.dll* and *playready.info* files.
6. Rebuild the whole browser to affect the changes using the command: Run *./mach build*

[30]https://dxr.mozilla.org/mozilla-central/source/

7. Implement the call-flow as that of EME as shown in the figure-34 by creating aMediaSession to send the samples from the GMP and calls the Media Foundation API[31] for decryption and decoding purposes.
8. With the session establishment, the identification of the PlayReady KeySystem is reflected at the browser console and with the information of the available CDM.

The figure-39 shows the architecture of the implementation of secure interface between the browser and the Microsoft Media Foundation. The Microsoft PlayReady being the DRM allows the samples to be decrypted and decoded using media foundation API's which are accessed from the SANDBOX'd .dlls residing in the browser. The samples are rendered and are ready to be painted. This phenomenon is adapted from the pipeline theory as illustrated in chapter 2.2.

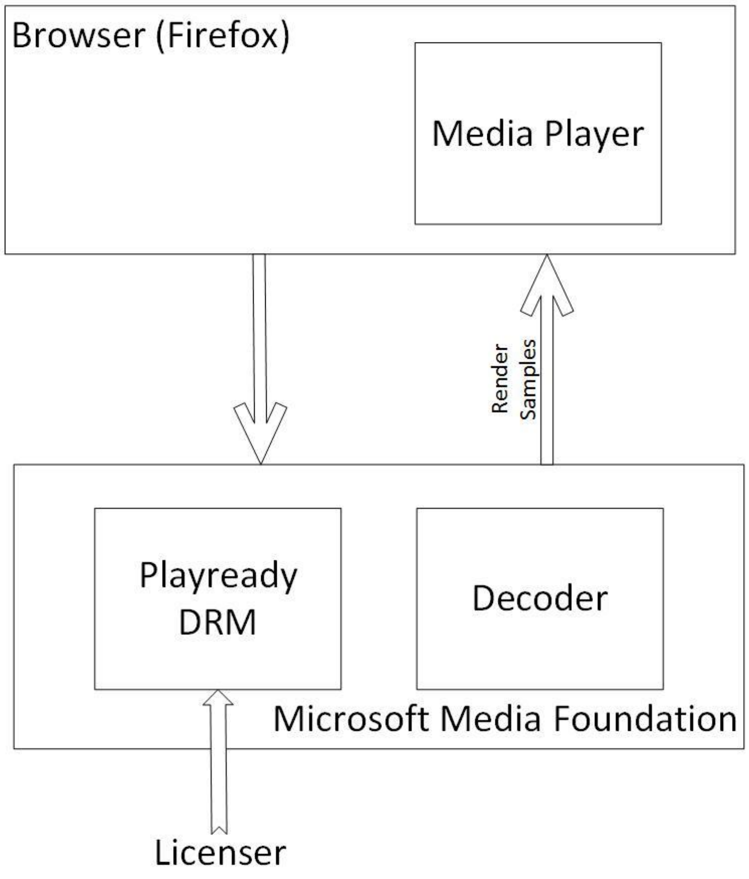


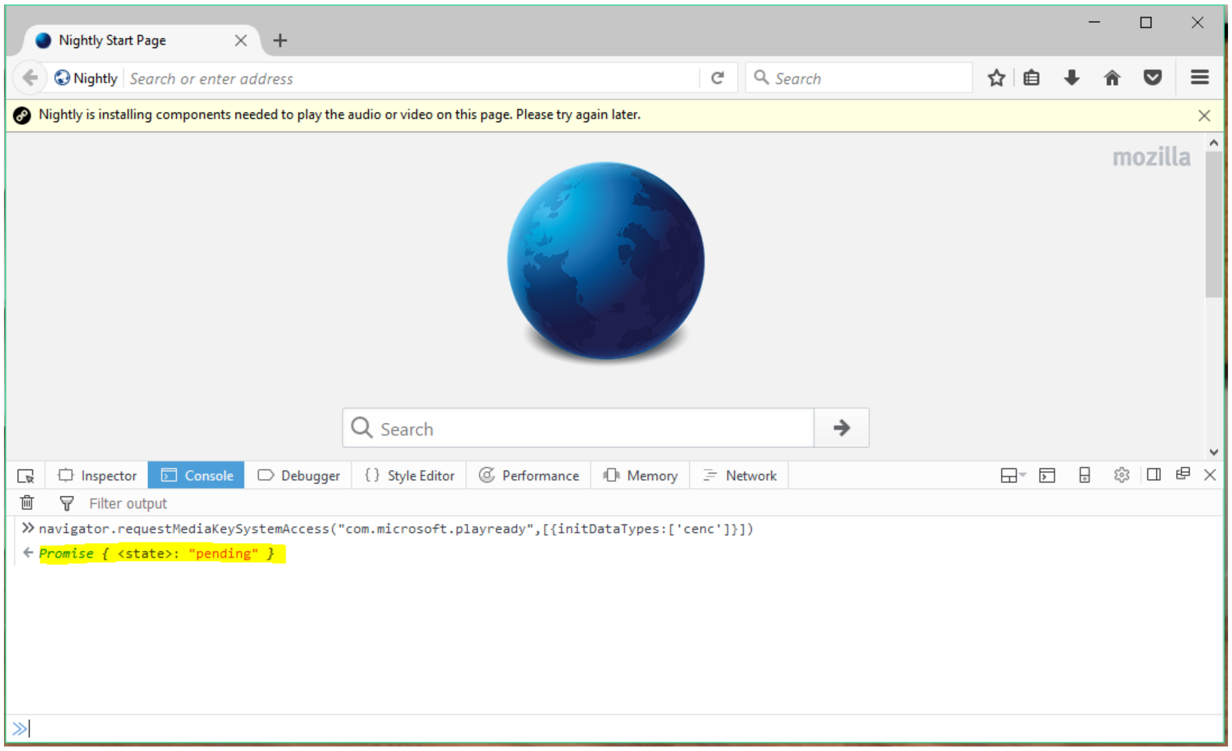
Figure 39: Call flow between browser and hardware-DRM

9. The PlayReady DRM that makes the playback of the decoded samples completely secure.
10. The decoded samples are rendered back to the browser wherein the media player is plugged in by creating a hole. This way, the rendered samples are played back in a secure way.

[31]https://msdn.microsoft.com/en-us/library/windows/desktop/ms697062(v=vs.85).aspx

In the current scenario, due to the changes related to the official migration by the MSE/EME developers of Mozilla browser, towards the Chromium Content Decryption Module (CDM) for the interface from the browser, has created a breakdown to the implementation of the secure Open CDM.

The Bugzilla[32] report that filters the files that affected the browser communication from the browser's CDM point of view is as shown in the Appendix C . The result of which is that the JavaScript fires the "pending" response.



```
>> navigator.requestMediaKeySystemAccess("com.microsoft.playready",[{initDataTypes:['cenc']}])
← Promise { <state>: "pending" }
```

Figure 40: JavaScript Response

Currently, the implementation is dependent on the bug-fix as mentioned earlier so that the browser establishes a call to the JavaScript upon identification of the available Chromium CDM. The JavaScript would return the status as "fulfilled" and the call flow is continued as per the EME specification.

With the successful response from the JavaScript layer, the KeySystem accesses the CDM and the samples will be sent for decryption. The samples then follow the call flow as shown in the figure-39. The decoded samples are painted on the media player which will be plugged into the browser which is logically possible by drilling a hole and this way, the samples are kept secure from the outside world.

[32]https://reviewboard.mozilla.org/r/98188/diff/10/

6. SECURITY ASPECTS

Security design is one of the challenging areas which require an effort to build a strong system with features that does not hinder the working process of a system under consideration. And this in particular is a concern when it comes to compromising between the owner's copyrights and the access rights and exchange and distribution of the information to the customers.

Over the course of the thesis progress, 3 notable technologies such as DRM, TEE and SANDBOX that provide the security and their usage on different platforms has been studied. Following subsections highlights these features.

6.1 DRM

An effective DRM technology must be designed with an intention to suffice the requirement of a smooth and effective handling of the copyrighted contents and must implement the secure method to prevent the unauthorized access of such contents. In practice, the current available DRM schemes such as Widevine from Google, PlayReady by Microsoft or FairPlay by Apple and some other schemes offer such secure approaches/mechanisms over different platforms.

The major security requirements that Microsoft PlayReady offers has been addressed here.

1. Controlled distribution and consumption of the copyrighted content:
   The license is issued to the digital content and the content is distributed based on the distribution rules in the license. Upon being distributed, the digital content would be consumed only with a valid license; otherwise, the content will be impossible to playback.
2. Robustness of the DRM scheme:
   - Protection policy: The storage and the distribution of the digital content must be protected by all means. The protection must be highly secure so as to ensure that even if the content is accessed by the third parties, it will not be consumed.
   - Strong bond between the content, license and the consumer:
     The license that is used for authorized access of the content must be logically associated with the content and the content owners must be able to validate the license.
   - Revoke: It must be able to revoke a device on a particular platform that has been fully compromised.

The DRM Usage and the license usage has been described in the figure-41:

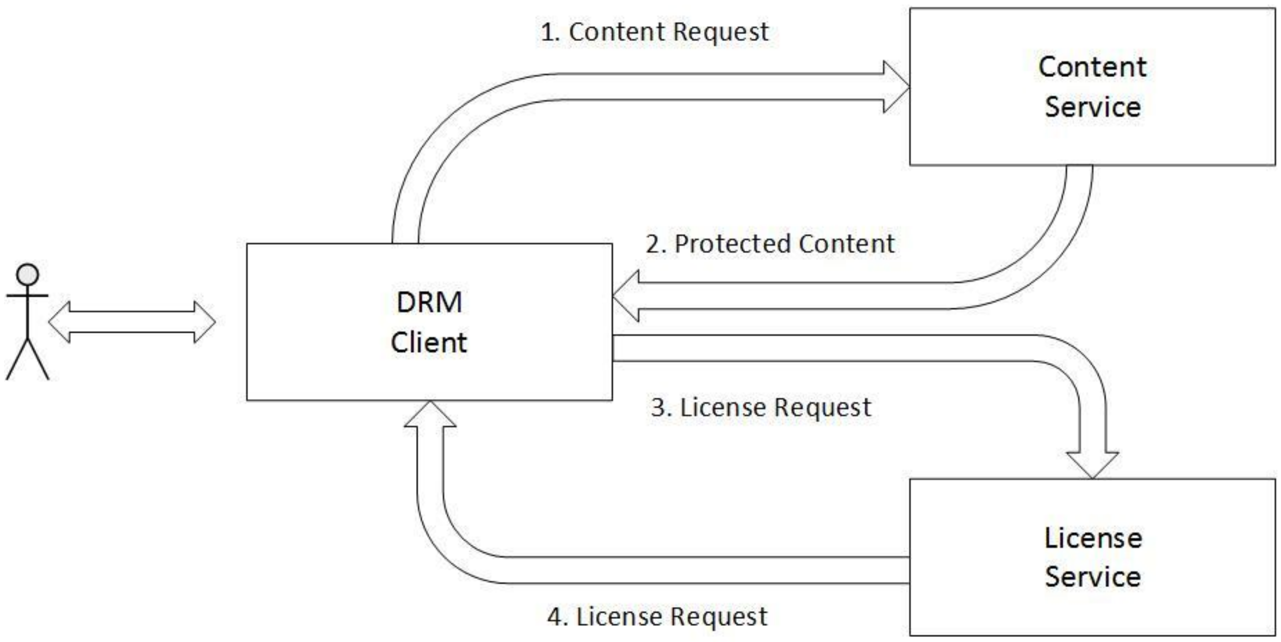


Figure 41: DRM usage

License acquisition and distribution: The consumer communicates with its DRM to get the protected content. The DRM sends a contact request to the Content service. The protected content is sent to the DRM client which then sends a request to the License Service which issues the license. Thus the DRM acquires the license and is distributed back to the consumer.

Content usage: The license is not only bound to the content protection but also to the legitimate consumer. Distribution of this license to other consumer or a third party attack to consume the content thus becomes useless. This scenario is presented in figure-42.

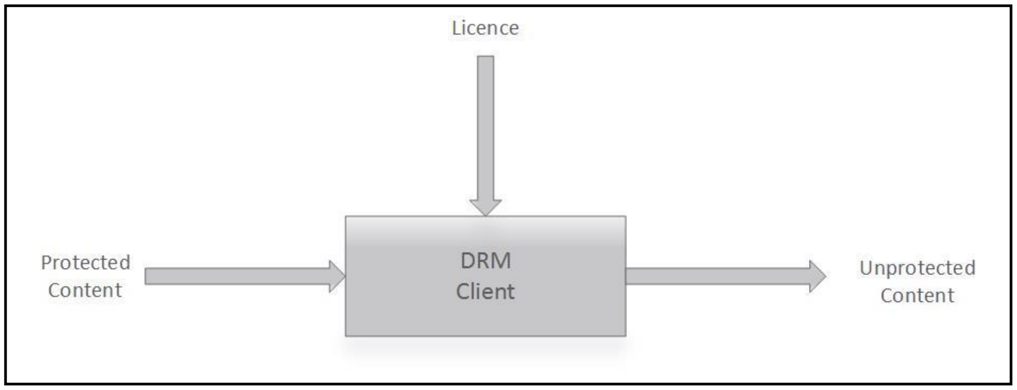


Figure 42: License Usage

The figure-43 below shows the significance of the protection involving different attributes such as Key ID, License key, Unprotected content, Consumer and the Protected Content.

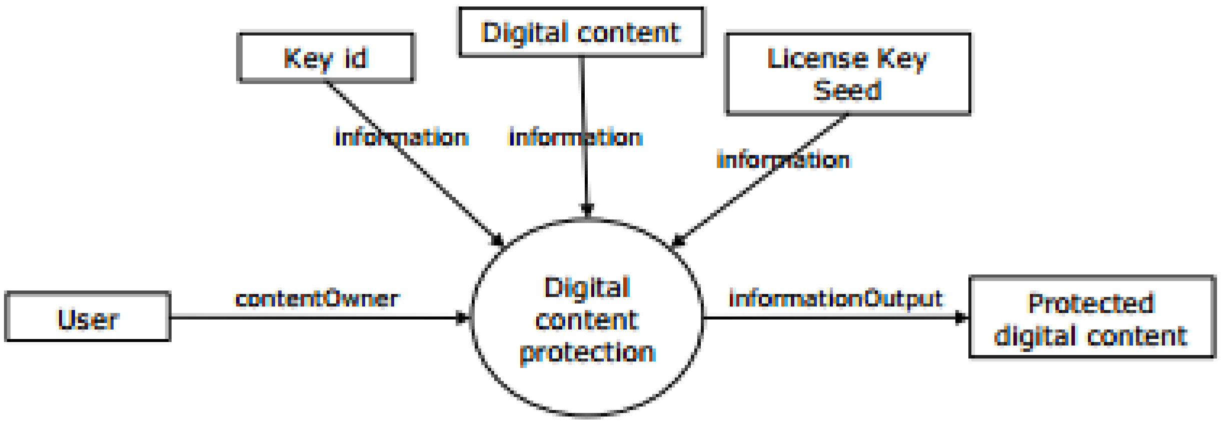


Figure 43: Process of Content protection

Microsoft PlayReady HW DRM

Following properties shows the advantages of considering HW assisted PlayReady DRM[33]:

1. Enables secure playback of high definition (HD) and ultra-high definition (UHD) content on multiple device platforms
2. Key material (such as private keys, content keys, and any other key material) and decrypted compressed and uncompressed video samples are protected by leveraging hardware security
3. Provides acquisition of multiple licenses in one message, thus preventing the delay for license acquisition when the user selects the content to play. This also enables the audio and video streams to be encrypted using separate keys, with which a single license acquires all the licenses requested within a content file.

For detailed security aspects of PlayReady DRM, please refer Microsoft PlayReady SL3000[34]

6.2 TEE

The TEE security functionalities provides the logical boundary between the Software and Hardware interfaces as explained in subchapters 4.2 and 4.3. The TEE instantiation through the secure boot process ensures the authenticity and acts as an asset to the integrity of the TEE. Below are a few security considerations that ensures the said secure isolation:

[33]https://docs.microsoft.com/en-us/windows/uwp/audio-video-camera/hardware-drm
[34]https://www.microsoft.com/playready/features/EnhancedContentProtection.aspx

- Trusted storage of the applications, TEE data and keys
- Ensuring confidentiality binding to the TEE
- Random number generator
- Cryptographic API for generating and deriving key pairs using algorithms such as SHA-256, AES 128/256, RSA 2048
- User Authentication
- Trusted Validation

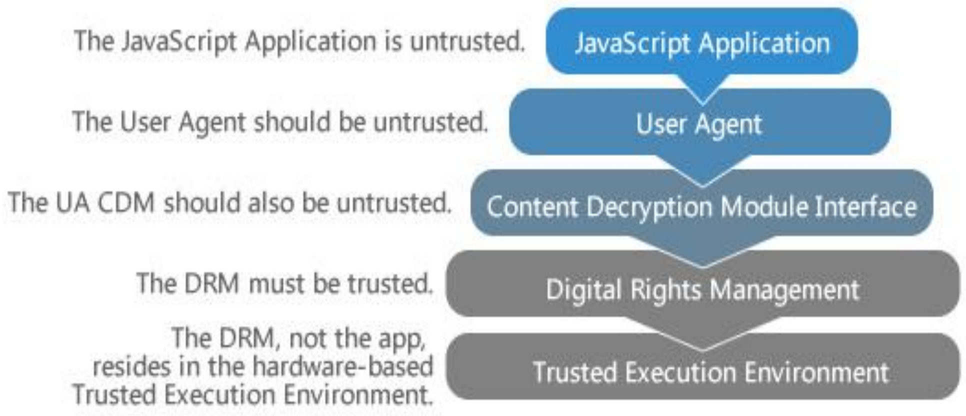

The JavaScript Application is untrusted.
JavaScript Application
The User Agent should be untrusted.
User Agent
The UA CDM should also be untrusted.
Content Decryption Module Interface
The DRM must be trusted.
Digital Rights Management
The DRM, not the app, resides in the hardware-based Trusted Execution Environment.
Trusted Execution Environment

Figure 44: CDM and TEE flow

6.3 SANDBOX

SANDBOX is considered as an important security feature as it can help to prevent to abuse a vulnerability that accesses the entire system. The SANDBOX security involves three levels; Level 0 is the least restrictive level, level 1 with certain restrictions, level 2 is the most restrictive currently. Once level 3 is introduced, it will become the most restrictive level.

Ideally for Chromium[35], the SANDBOX would force the rendering engine to use the kernel API to interact with the outside world. Currently, the Chromium relies on Windows specific features to SANDBOX the rendering engine. And the implementation of the OCDM is done on a Linux platform, the FAT32 file system does not support access control lists. And without this, the windows security manager. As a result, the implementation of the PlayReady premium videos for playback is insecure. As a result, to make the playback more secure, the concept of TEE has been introduced in the chapter as an additional implementation.

[35] https://www.chromium.org/developers/design-documents/sandbox

The Firefox[36]browser comes along with a SANDBOX’ed multi-process architecture that provides enhanced browser's security. Firefox’s SANDBOX model allows 3 different levels[37]. Currently, the parent Firefox browser configures;

- ALL: Gecko Media Plug-in enabled
- Windows: NPAPI Plug-in enabled, content at level 2, other channels at level 1 and compositor at level 0.
- OSX: content at level 2
- Linux: content at level 2

Further, the Firefox browser is aiming to have level 3 SANDBOX'ing for Windows in its future releases.

[36] https://wiki.mozilla.org/Security/Sandbox
[37] https://dxr.mozilla.org/mozilla-central/source/security/sandbox/chromium/sandbox/win/src/security_level.h

7. CONCLUSION AND ENHANCEMENT

The main goal of the thesis was to design and implement a secure interface between the open content decryption module and hardware-assisted DRM and this has realized on Chromium and Mozilla Firefox browsers using the existing github open source CDM project of Fraunhofer FOKUS. The design approach for both of these browsers has been different, the former using the CDMi specification set by software-based Microsoft Playready Porting kit on Linux-Ubuntu platform and the latter using the hardware-based PlayReady DRM on Windows 10 platform.

However, it has been evident that, without the SANDBOX security, the Chromium implementation of enabling the playback of the PlayReady videos was not secure. And hence, a possibility of TEE implementation has been studied and the design approach to implement the TEE architecture is presented in this thesis as the future scope to ensure the secure playback of PlayReady DASH videos. In case of Firefox, the browser allowed a set of Media foundation DLL's within its SANDBOX mechanism which enabled the samples to communicate to the Media Foundation PlayReady DRM and thus creating a secure interface. However, with the migration of GMP decryption module to Chromium Content Decryption Module as mentioned in section so and so needs further implementation in building the playready.dll.

The implementation of TEE under both Chromium and Firefox browsers along with the level 3 SANDBOX technology would make the browsers ideal for the playback of the premium protected videos. This would be the driving force to step an inch closer to implement the next generation high definition videos such as 8K, 360°videos.

8. REFERENCES

[1] "Motion Picture Laboratories, Inc. (2013) MovieLabs Specification forNext Generation Video – Version 1.0," MovieLabs, [Online]. Available: http://www.movielabs.org/ngvideo/MovieLabs%20Specification%20for%20Next%20Generation%20Video%20v1.0.pdf. [Accessed 20 02 2017].

[2] "Microsoft Playready - Developing Playready Clients," Microsoft, 2015.

[3] "Overview of the Media Foundation Architecture," Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/ff819455(v=vs.85).aspx.

[4] "Overview of the Media Foundation Architecture," Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/ff819455(v=vs.85).aspx.

[5] R. a. M. E. W. Pantos, "HTTP Live Streaming," IETF, [Online]. Available: http://tools.ietf.org/html/draft-pantos-http-live-streaming-06, March 2011. [Accessed 24 Feb 2017].

[6] "IIS Smooth Streaming Transport Protocol," Microsoft, [Online]. Available: http://www.iis.net/community/files/media/smoothspecs/[MS-SMTH].pdf, September 2009. [Accessed 24 Feb 2017].

[7] "Adobe HTTP Dynamic Streaming," Adobe, [Online]. Available: http://www.adobe.com/products/httpdynamicstreaming/. [Accessed 24 Feb 2017].

[8] "Media Source Extensions," W3C, [Online]. Available: https://www.w3.org/TR/2014/CR-media-source-20140717/.

[9] "Encrypted Media Extensions," W3C, [Online]. Available: https://www.w3.org/TR/2016/CR-encrypted-media-20160705/.

[10] I. Cable Television Laboratories, "HTML 5 Premium Media Extensions," 2014.

[11] "Common encryption in ISO base media file format files," International Organization for Standardization, 2016.

[12] "Device Specifications," Global Platform, [Online]. Available: http://www.globalplatform.org/specificationsdevice.asp.

[13] G. Inc., "TEE Protection Profile," 2014.

[14] "Trusted Platform Module (TPM) Specifications," [Online]. Available: https://trustedcomputinggroup.org/tpm-main-specification. [Accessed 20 02 2017].

[15] "The Chromium Projects," [Online]. Available: https://www.chromium.org/.

[16] "Inter-process Communication (IPC)," [Online]. Available: https://www.chromium.org/developers/design-documents/inter-process-communication.

[17] "Multi-process Architecture," [Online]. Available: https://www.chromium.org/developers/design-documents/multi-process-architecture.

[18] "How Chromium Displays Web Pages," [Online]. Available: https://www.chromium.org/developers/design-documents/displaying-a-web-page-in-chrome.

[19] "The Chromium Projects," [Online]. Available: http://www.chromium.org/nativeclient/getting-started/getting-started-background-and-basics.

[20] "Pepper plugin implementation," [Online]. Available: https://sites.google.com/a/chromium.org/dev/developers/design-documents/pepper-plugin-implementation#TOC-Architecture-of-the-backend-implementation.

[21] "Encrypted Media Extensions," W3C, [Online]. Available: https://www.w3.org/TR/2016/CR-encrypted-media-20160705/.

[22] "Architecture notes OCDM," Fraunhofer FOKUS, [Online]. Available: https://github.com/fraunhoferfokus/open-content-decryption-module/blob/2357_ppapi_chromium/docs/architecture_notes_ocdm.md.

[23] J. C. S. a. S.Arbanowski., "Interoperability, Digital Rights Management and The Web," [Online].

[24] S. A. S. K. S Pham, "An Open Source Content Decryption Module to Improve DRM Integration with HTML5 Platforms," in *IEEE International Symposium on Multimedia (ISM) 2015, 417-420*.

[25] "PMP Media Session," Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/gg153558(v=vs.85).aspx.

[26] "About Topologies," Microsoft, [Online]. Available: https://msdn.microsoft.com/de-de/library/windows/desktop/aa369306(v=vs.85).aspx.

[27] "GeckoMediaPlugins," Mozilla, [Online]. Available: https://wiki.mozilla.org/GeckoMediaPlugins.

9. APPENDIX

9.1 Appendix A

APIs Mapping from MSE in JavaScript to Media Foundation MediaEngine in C++

JavaScript (MSE)	MFMediaEngine C++
[Constructor]	IMFMediaEngineClassFactoryEx::CreateMediaSourceExtension
MediaSource	IMFMediaSourceExtension
MediaSource.sourceBuffers	IMFMediaSourceExtension::GetSourceBuffers
MediaSource.activeSourceBuffers	IMFMediaSourceExtension::GetActiveSourceBuffers
MediaSource.readyState	IMFMediaSourceExtension::GetReadyState
MediaSource.duration	IMFMediaSourceExtension::GetDuration IMFMediaSourceExtension::SetDuration
MediaSource.addSourceBuffer	IMFMediaSourceExtension::AddSourceBuffer
MediaSource.removeSourceBuffer	IMFMediaSourceExtension::RemoveSourceBuffer
MediaSource.endOfStream	IMFMediaSourceExtension::SetEndOfStream
MediaSource.isTypeSupported	IMFMediaSourceExtension::IsTypeSupported

SourceBuffer	IMFSourceBuffer
SourceBuffer.updating	IMFSourceBuffer::GetUpdating
SourceBuffer.buffered	IMFSourceBuffer::GetBuffered
SourceBuffer.timestampOffset	IMFSourceBuffer::GetTimeStampOffset IMFSourceBuffer::SetTimeStampOffset
SourceBuffer.audioTracks	No equivalent in Media Foundation
SourceBuffer.appendWindowStart;	IMFSourceBuffer::GetAppendWindowStart IMFSourceBuffer::SetAppendWindowStart
SourceBuffer.appendWindowEnd	IMFSourceBuffer::GetAppendWindowEnd IMFSourceBuffer::SetAppendWindowEnd
SourceBuffer.appendBuffer(ArrayBuffer data)	IMFSourceBuffer::Append
SourceBuffer.appendBufferArrayBufferView data)	IMFSourceBuffer::Append
SourceBuffer.appendStream	IMFSourceBuffer::AppendByteStream
SourceBuffer.abort	IMFSourceBuffer::Abort
SourceBuffer.remove	IMFSourceBuffer::Remove

SourceBufferList	IMFSourceBufferList
SourceBufferList.length	IMFSourceBufferList::GetLength
getter SourceBuffer	IMFSourceBufferList::GetSourceBuffer
VideoPlaybackQuality	IMFMEdiaEngineEx::GetStatistics
VideoPlaybackQuality.creationTime	No Media Foundation equivalent
VideoPlaybackQuality.totalVideoFrames	MF_MEDIA_ENGINE_STATISTIC_FRAMES_RENDERED + MF_MEDIA_ENGINE_STATISTIC_FRAMES_DROPPED These flags are defined in MF_MEDIA_ENGINE_STATISTIC
VideoPlaybackQuality.droppedVideoFrames	MF_MEDIA_ENGINE_STATISTIC_FRAMES_DROPPED This flag is defined in MF_MEDIA_ENGINE_STATISTIC.
VideoPlaybackQuality.totalFrameDelay	MF_MEDIA_ENGINE_STATISTIC_TOTAL_FRAME_DELAY This flag is defined in MF_MEDIA_ENGINE_STATISTIC.
DOMString URL.createObjectURL(MediaSource mediaSource)	No equivalent in Media Foundation
HTMLVideoElement .getVideoPlaybackQuality	See VideoPlaybackQuality notes above

AudioTrack.kind	No Media Foundation equivalent
AudioTrack.language	IMFMediaEngineEx::GetStreamAttribute Note, pass MF_SD_LANGUAGE as the value for the *guidMFAttribute* parameter.
AudioTrack.sourceBuffer	IMFMediaSourceExtension::GetSourceBuffer
MediaSource.sourceopen	IMFMediaSourceExtensionNotify
MediaSource.sourceended	IMFMediaSourceExtensionNotify::OnSourceEnded
MediaSource.sourceclose	IMFMediaSourceExtensionNotify::OnSourceClose
SourceBuffer.updatestart	IMFSourceBufferNotify::OnUpdateStart
SourceBuffer.update	IMFSourceBufferNotify::OnUpdate
SourceBuffer.updateend	IMFSourceBufferNotif::OnUpdateEnd
SourceBuffer.error	IMFSourceBufferNotify::OnError
SourceBuffer.abort	IMFSourceBufferNotify::OnAbort
SourceBufferList.addsourcebuffer	IMFBufferListNotify::OnAddSourceBuffer
SourceBufferList.removesourcebuffer	IMFBufferListNotify::OnRemoveSourceBuffer

9.2 Appendix B

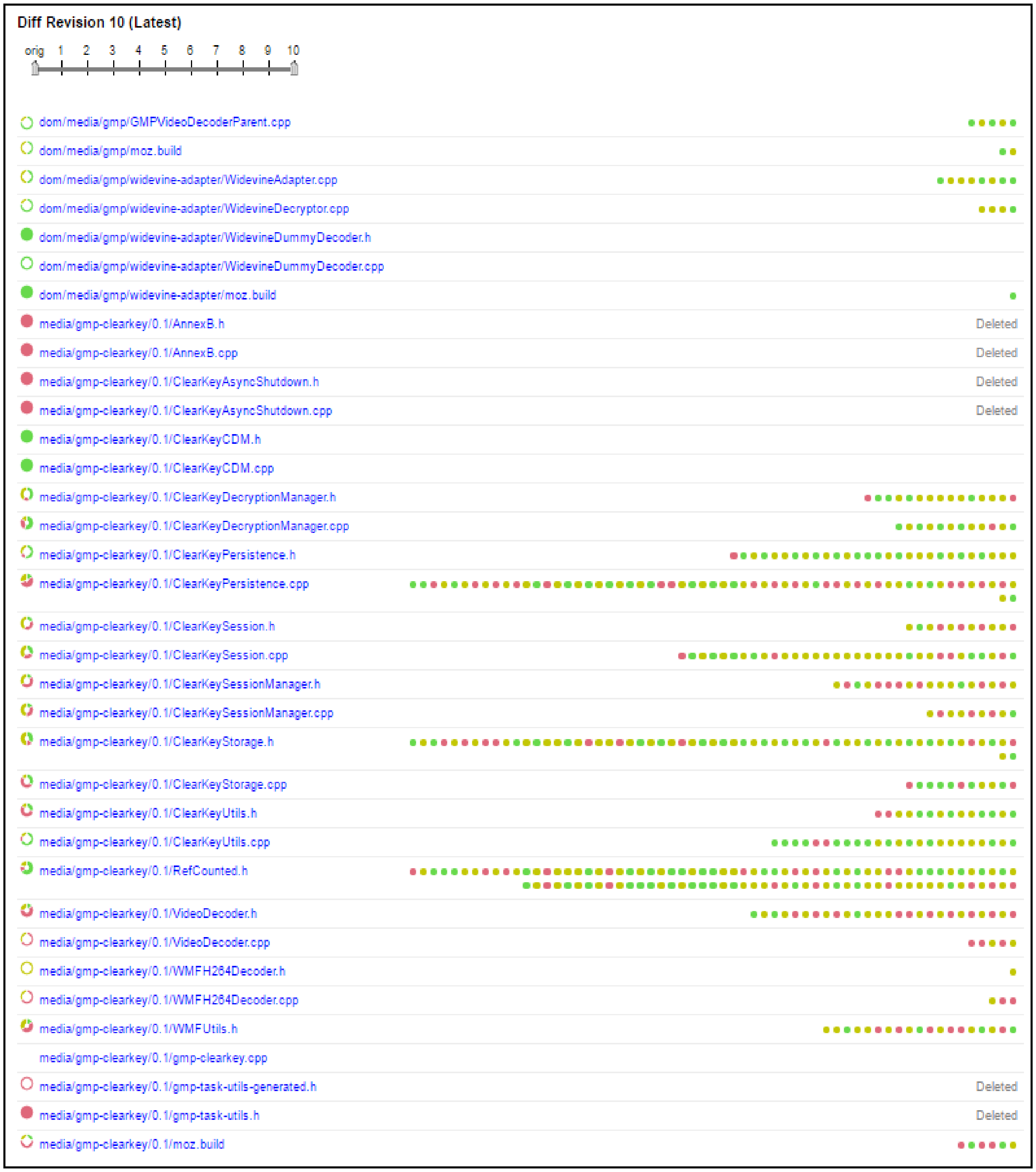
APIs Mapping from EME in JavaScript to Media Foundation MediaEngine in C++

JavaScript (MSE)	MFMediaEngine C++
HTMLMediaElement.msKeys	IMFMediaEngineEME::get_Keys
HTMLMediaElement.msSetMediaKeys	IMFMediaEngineEME::SetMediaKeys
onmsneedkey	
[Constructor]	IMFMediaEngineClassFactory2::CreateMediaKeys2
MSMediaKeys	IMFMediaKeys
MSMediaKeys.keySystem	IMFMediaKeys::get_KeySystem
MSMediaKeys.createSession	IMFMediaKeys::CreateSession
MSMediaKeys.isTypeSupported	IMFMediaEngineClassFactoryEx::IsTypeSupported
	IMFMediaKeys::GetSuspendNotify
	IMFMediaKeys::Shutdown
MediaKeySession	IMFMediaKeySession
MediaKeySession.error	IMFMediaKeySession::GetError
MediaKeySession.keySystem	IMFMediaKeySession::get_KeySystem

MediaKeySession.sessionId	IMFMediaKeySession::get_SessionId
MediaKeySession.update	IMFMediaKeySession::Update
MediaKeySession.close	IMFMediaKeySession::Close
HTMLSourceElement.keySystem	IMFMediaEngineSrcElementsEx::GetKeySystem
Event interface additions	
MediaKeyMessageEvent	Equates to IMFMediaKeySessionNotify
MediaKeyMessageEvent.message	
MediaKeyMessageEvent.destinationURL	
MediaKeyNeededEvent	Equates to IMFMediaEngineNeedKeyNotify
MediaKeyNeededEvent.initData	
Events	
Keyadded	IMFMediaKeySessionNotify::KeyAdded
Keyerror	IMFMediaKeySessionNotify::KeyError
Keymessage	IMFMediaKeySessionNotify::KeyMessage
Msneedkey	IMFMediaEngineNeedKeyNotify::NeedKey

9.3 Appendix C

Bugzilla report – Migration towards Chromium CDM



Diff Revision 10 (Latest)

dom/media/gmp/GMPVideoDecoderParent.cpp
dom/media/gmp/moz.build
dom/media/gmp/widevine-adapter/WidevineAdapter.cpp
dom/media/gmp/widevine-adapter/WidevineDecryptor.cpp
dom/media/gmp/widevine-adapter/WidevineDummyDecoder.h
dom/media/gmp/widevine-adapter/WidevineDummyDecoder.cpp
dom/media/gmp/widevine-adapter/moz.build
media/gmp-clearkey/0.1/AnnexB.h Deleted
media/gmp-clearkey/0.1/AnnexB.cpp Deleted
media/gmp-clearkey/0.1/ClearKeyAsyncShutdown.h Deleted
media/gmp-clearkey/0.1/ClearKeyAsyncShutdown.cpp Deleted
media/gmp-clearkey/0.1/ClearKeyCDM.h
media/gmp-clearkey/0.1/ClearKeyCDM.cpp
media/gmp-clearkey/0.1/ClearKeyDecryptionManager.h
media/gmp-clearkey/0.1/ClearKeyDecryptionManager.cpp
media/gmp-clearkey/0.1/ClearKeyPersistence.h
media/gmp-clearkey/0.1/ClearKeyPersistence.cpp
media/gmp-clearkey/0.1/ClearKeySession.h
media/gmp-clearkey/0.1/ClearKeySession.cpp
media/gmp-clearkey/0.1/ClearKeySessionManager.h
media/gmp-clearkey/0.1/ClearKeySessionManager.cpp
media/gmp-clearkey/0.1/ClearKeyStorage.h
media/gmp-clearkey/0.1/ClearKeyStorage.cpp
media/gmp-clearkey/0.1/ClearKeyUtils.h
media/gmp-clearkey/0.1/ClearKeyUtils.cpp
media/gmp-clearkey/0.1/RefCounted.h
media/gmp-clearkey/0.1/VideoDecoder.h
media/gmp-clearkey/0.1/VideoDecoder.cpp
media/gmp-clearkey/0.1/WMFH264Decoder.h
media/gmp-clearkey/0.1/WMFH264Decoder.cpp
media/gmp-clearkey/0.1/WMFUtils.h
media/gmp-clearkey/0.1/gmp-clearkey.cpp
media/gmp-clearkey/0.1/gmp-task-utils-generated.h Deleted
media/gmp-clearkey/0.1/gmp-task-utils.h Deleted
media/gmp-clearkey/0.1/moz.build

9.4 Appendix D

GITHUB project links:

Chromium	https://github.com/RohanKrishnamurthy/Chromium
Firefox-Nightly	https://github.com/RohanKrishnamurthy/Firefox-Nightly
Microsoft MF Media Player	https://github.com/RohanKrishnamurthy/Media-Foundation-Media-Player
TEE	https://github.com/RohanKrishnamurthy/TEE

10. GLOSSARY

A

ABR	Adaptive Bit Rate (14)
AES	Advanced Encryption Standard (9,59)
API	Application Programming Interface (9, 12,15)
AAC	Advanced Audio Coding (46)

C

CDM	Content decryption module (3, 26,30-35,46,48-50,53,54,55,59)
CDMI	Content decryption module interface (4,31,32,33,62)
CDN	Content Delivery Network (9,14)
CENC	Common Encryption (2,16)

D

DASH	Dynamic Adaptive Streaming over HTTP (2, 14,16,62)
DRM	Digital rights Management (2,3,4-8, 12,14,15,16,20,29,30,31,32,37,38,45,47,53,56,57,58,62)

E

EME	Encrypted Media Extensions (12,14,15,16,29,30,31,45,46,48,49,53,54)

G

GMP	Gecko Media Plug-in (4,12,45,48,49,50,52,53,62)

H

HD	High Definition (2)
HDCP	High-bandwidth Digital Content Protection
HEVC	High Efficiency Video Coding (3)
HFR	High frame Rate (3)
HLS	HTTP Live Streaming (14)
HTML	Hyper Text Markup Language (3)
HTTP	Hypertext Transfer Protocol (14

I

IETF	Internet Engineering Task Force (14
I/O	Input/output
IP	Internet Protocol
IPC	Inter-process Communication (25,26,27,30)

M

MF	Media Foundation (10,11,38,39,45)
MFT	Media Foundation Transform (11)
MPEG	Moving Picture Experts Group (2,14,16)
MSE	Media Source Extensions (14,15,16,45,47,54)

N

NPAPI	Netscape Plug-in Application Programming Interface (27,60)

O

OCDM	Open-source Content decryption Module (3,29,30,32-35,59)
OCDMI	Open-source Content decryption Module interface (32-35)
OMA-DRM	Open Mobile Alliance – DRM (6)
OMA-SCE	Open Mobile Alliance -Secure Content Exchange (6)
OTT	Over-the-top (2)

P

PE	Protected Environment (37)
PPAPI	Pepper Plug-in API (4,25,27,29,30,34)
PMP	Protected Media Path (37 – 40)
PSSH	Protection System Specific Header (16)

R

REE	Rich Execution Environment (18, 19)
RISC	Reduced instruction Set Computing (22,23)
RPC	Remote Procedure Call (30,31,32,35)
RTP	Real-time Transport Protocol (14)

S

SE	Secure Environment (18)
SHA	Secure Hash Algorithm (59)

T

TA	Trusted Application (4,7,18,19,21)
TEE	Trusted Execution Environment (4,18,19,20,21,22,23,56,58,59,62)
TPM	Trusted Platform Module (21)

U

UHD	Ultra High Definition (58)
URL	Uniform resource Locator (11,15,39,52)

V

VS	Visual Studio (47)

W

W3C	World Wide Web Consortium (14,15,16,29,31,48)
WMDRM	Windows Media DRM (9)