Secure Remote Service Execution for Web Media Streaming

vorgelegt von
Dipl.-Ing. Alexandra Mikityuk
geb. in Leningrad, UdSSR

von der Fakultät IV – Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
- Dr.-Ing. -

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender:	Prof. Dr. Thomas Magedanz, Technische Universität Berlin
Gutachter:	Prof. Dr. Jean-Pierre Seifert, Technische Universität Berlin
Gutachter:	Prof. Dr. Jean-Claude Dufourd, ParisTech
Gutachter:	Prof. Dr.-Ing. Ina Schieferdecker, Technische Universität Berlin

Tag der wissenschaftlichen Aussprache: 29. August 2017

Berlin 2017
D 83

Abstract

Through continuous advancements in streaming and Web technologies over the past decade, the Web has become a platform for media delivery. Web standards like HTML5 have been designed accordingly, allowing for the delivery of applications, high-quality streaming video, and hooks for interoperable content protection.

Efficient video encoding algorithms such as AVC/HEVC and streaming protocols such as MPEG-DASH have served as additional triggers for this evolution. Users now employ Web browsers as a tool for receiving streaming media and rendering Web applications, and browsers have been embedded into almost every kind of connected device.

The drawback of these technical developments and quick rate of user adoption is that modern Web browsers have introduced significant constraints on devices' capabilities. First, the computational requirements have risen continuously, resulting in a cycle where modern devices can be nearly outdated after a year or two. Second, as the integration of browser technologies is a complicated matter, not every platform provides the same performance. Different Operating Systems (OSs), chipsets and software engines are the main reasons for this difference in performance.

The result is a tremendous diversity of devices, which demands significant efforts in application development and requires platform-specific adaptions. Finally, browsers, with their prominence on the Web, have become an attractive weak point for hackers to attack.

This thesis addresses the drawbacks of local browser execution with a novel concept for Secure Remote Service Execution (SRSE). SRSE does not treat the symptoms of the problem. Instead, it addresses the cause. By shifting application execution to a remote machine, it delivers the entire user interface as a video stream to end-users. This resolves device dependencies and enables a secure execution, as none of the application code is executed on the device.

This thesis also defines various architectural approaches to SRSEs. They enable flexibility with regard to addressing traditional media services with SRSEs and can create significant computing and network resource savings.

As state-of-the-art Web media delivery is based on locally executed services, SRSEs have a profound impact on existing standards and corresponding implementations. As a response to these environments, new techniques and interfaces are identified and specified throughout this thesis as the need for them arises.

Zusammenfassung

Im letzten Jahrzehnt hat sich das Internet durch kontinuierliche Fortschritte im Bereich der Streaming- und Web-Technologien zu einer Plattform für die Videobereitstellung entwickelt. Entsprechend wurden Web-Standards wie HTML5 eingeführt, welche die Bereitstellung von Anwendungen, qualitativ hochwertigem Video-Streaming und die Integrationen von interoperablen Inhalte-Schutz-Mechanismen ermöglichen.

Neben effizienten Video-Kodierungsalgorithmen dienten moderne Streaming-Protokolle als zusätzliche Auslöser für diese Entwicklung. Für den Anwender ist der Web-Browser zum universellen Werkzeug für den Empfang von Video und dem Rendering von Web-Anwendungen geworden und ist heute in fast alle Arten von internetfähigen Geräten integriert.

Der Nachteil dieser sehr schnellen technischen Entwicklungen und großen Benutzerakzeptanz ist, dass moderne Web-Browser erhebliche und ständig steigende Anforderungen an die Fähigkeiten der Geräte stellen. Erstens sind die Rechenanforderungen kontinuierlich gestiegen, was zu einem Zyklus führte, in dem moderne Geräte bereits nach zwei Jahren ersetzt werden müssen. Zweitens - und speziell aufgrund der Komplexität bei der Integration von Browser-Technologie in die Endgeräte - bietet nicht jede Plattform die gleiche Leistung: unterschiedliche Betriebssysteme, Chipsätze und Software-Treiber sind die wichtigsten Gründe für Leistungsunterschiede.

Hiermit ergibt sich eine enorme Vielfalt von Geräten, die erhebliche Anstrengungen bei der Anwendungsentwicklung verlangen und oft plattformspezifische Anpassungen erfordern. Schließlich ist der Browser durch seine Bedeutung im Web und bekannte Schwachstellen zu einem attraktiven Ziel für Hacker geworden.

Die vorliegende Arbeit adressiert die soeben beschriebenen Nachteile der lokalen Ausführung von Web-Applikationen mit der Einführung eines neuartigen Konzepts, genannt Secure-Remote-Service-Execution (SRSE). SRSE behandelt nicht die Symptome der lokalen Applikationsausführung, sondern befasst sich mit der Ursache: Durch das Verlagern der Ausführung der Web-Anwendungen auf einen entfernten Rechner liefert SRSE ein Abbild der Applikation als Videostrom zum Endnutzer. Dies löst das Problem der Abhängigkeit vom Endgerät und ermöglicht hiermit eine sichere, in der Cloud gekapselte Dienstausführung.

Im Rahmen der Arbeit werden verschiedene Architekturansätze für SRSEs definiert. Sie ermöglichen Flexibilität bei der Adaptierung von traditionellen Mediendiensten für SRSEs und schaffen erhebliche Einsparungen von Rechen- und Netzwerkressourcen.

Durch die Einführung der SRSE ergibt sich außerdem ein nachhaltiger Anpassungsbedarf an die bisherigen Spezifikationen, Protokolle und Schnittstellen zur lokalen Applikationsausführung. Die neuen Anforderungen werden in der vorliegenden Arbeit analysiert, bewertet und anschließend in Form von neuen Verfahren und Schnittstellen definiert und teilweise prototypisch implementiert und evaluiert.

Acknowledgement

I am very grateful to my advisor Prof. Dr. Jean-Pierre Seifert for his knowledge and assistance, as well as motivating and encouraging me through all the years of my research.

I would also like to thank my supervisors Prof. Dr.-Ing. Ina Schieferdecker and Prof. Dr. Jean-Claude Dufourd for their very thorough feedback and insightful comments. I am indebted to my colleagues Dr. Oliver Friedrich, Dr. Stefan Arbanowski and Dr. Randolph Nikutta for their constant support, valuable feedback and generous guidance. Their immense knowledge was essential in building my understanding of this field of research.

My special thanks go out to my lab partners Stefan Pham, Benjamin Zachey, David Livshits, Eugen Osiptschuk, Robin Stinder and Dr. Julian Vetter for contributing to my work, supporting with their experience and our effective and fun collaboration.

I want to thank my family and friends for taking on the sometimes difficult task of consistently supporting me throughout all the PhD years, for their patience and for constantly motivating me.

To Jocky for his incredible support and endless belief in me.

Alexandra Mikityuk

Contents

List of Figures

List of Tables

List of Abbreviations

ABR	Adaptive Bitrate
AES	Advanced Encryption Standard
AGS	Aggregation Switch
AIT	Application Information Table
API	Application Programming Interface
CA	Conditional Access
CDM	Content Decryption Module
CDMi	Content Decryption Module Interface
CDN	Content Distribution Network
CEK	Content Encryption Key
CENC	ISO Common Encryption
CPE	Customer Premises Equipment
CPU	Central Processing Unit
CSS	Cascading Style Sheets
CTA	Consumer Technology Association
DASH	MPEG-Dynamic Adaptive Streaming over HTTP
DEE	Dual Execution Environment
DOM	Document Object Model
DRM	Digital Rights Management
DSLAM	Digital Subscriber Line Access Multiplexer
DSM-CC	Digital Storage Media Command and Control
DTV	Digital TV
DVB	Digital Video Broadcasting
EL	Exception Level
EME	Encrypted Media Extensions
EPG	Electronic Program Guide
HA	Host Application
HbbTV	Hybrid Broadcast Broadband TV
HD	High Definition
HDMI	High Definition Multimedia Interface
HEVC	High Efficiency Video Coding
HLS	HTTP Live Streaming
HTML	Hyper Text Markup Language
IP	Internet Protocol
JS	JavaScript
MHP	Multimedia Home Platform
MSE	Media Source Extensions
MSEE	Media Streaming Execution Environment
MUI	Media User Interface

NE	Network Element
OIPF	Open IPTV Forum
OS	Operating System
OTT	Over-The-Top
PAT	Program Association Table
PID	Packet ID
PKI	Public Key Infrastructure
PMT	Program Map Table
QoS	Quality of Service
RAM	Random Access Memory
RPC	Remote Procedure Call
RSE	Remote Service Execution
RTE	Runtime Environment
RTSP	Real Time Streaming Protocol
SCB	Secure Cloud Browser
SCHB	Secure Cloud HbbTV Browser
SD	Standard Definition
SES	Secure Streaming
SIC	Secure Interactive Contents
SK	Separation Kernel
SMC	Secure Monitor Call
SoA	State-of-the-Art
SoC	System on a Chip
SRSE	Secure Remote Service Execution
STB	Set-top Box
TA	Trusted Application
TCB	Trusted Computing Base
TCG	Trusted Computing Group
TEE	Trusted Execution Environments
TF	Task Force
TPM	Trusted Platform Module
TS	Transport Stream
TZ	TrustZone
UHD	Ultra High Definition
UI	User Interface
URL	Uniform Resource Locator
VM	Virtual Machine
VOD	Video on Demand
W3C	World Wide Web Consortium
WAVE	Web Application Video Ecosystem
WeSM	Weighted Scoring Model
WMS	Web Media Streaming
XHR	XML HTTP Request
XML	Extensible Markup Language

1 Introduction

> *Fragmentation is the evil twin of differentiation, a term marketing managers use to explain the need to create so many different handsets [A. 04].*
>
> **— Allen Lau**
> (Co-founder and CEO of Wattpad)

1.1 Motivation

Media streaming is currently the biggest growth area on the Web. According to the *Cisco VNI Forecast and Methodology, 2015-2020*, "*by 2019, 80% of global Internet consumption will be video content. It would take an individual over 5 million years to watch the amount of video that will cross global IP networks every single month in 2019. Over-The-Top (OTT) streaming will fuel the growth, as ultra-high definition 4K video becomes the new standard for users.*" [CIS14].

The growth of Web Media Streaming (WMS) platforms such as Netflix [Net17a] and Amazon Video [Ama17] confirms these predictions. In addition, services such as Facebook [Fac17], Instagram [Ins17], Twitter [Twi17], etc. are also increasing their volume of video through the continuous introduction of live and user-generated media content.

Currently, the most important differentiating factor for WMS providers is the Quality of Service (QoS) at the application level, namely the *Media User Interface (MUI)*. To address users' demands, WMS providers try to achieve a qualitatively high-level and consistent MUI while delivering to a wide range of user devices [Rad15] [BH06].

A multitude of devices implies a variety of different execution environments for media applications, which creates a highly fragmented device market for media service delivery. The impact of the fragmented device market on WMS is analyzed in the following through a demonstration of the most significant constraints caused by the fragmentation.

The ongoing convergence of the Web with media has made the Web browser the ultimate cross-platform *Media Service Execution Environment (MSEE*[1]*)*. Web browsers have improved in recent years, evolving from simple HyperText Markup Language (HTML) viewers to powerful, rich MSEEs with media, communication and security capabilities.

In this generation of browsers, the MUI is a Web application and is called a *browser-based MUI*. This approach is supported by various standardization organizations such as Hybrid Broadcast Broadband TV (HbbTV) [Hyb16], Consumer Technology Association (CTA) [CTA17], ITU Telecommunication Standardization Sector (ITU-T) [ITU17] and represents the current state-of-the-art in MSEE technology.

[1] MSEEs might include but are not limited to: application managers, runtime environments, libraries and provider-specific data.

Smart TVs, HbbTV and Google Chromecast [Goo17g], as well as service operators, OTT and PayTV providers have already commercialized browser-based MUIs [M. 13].

In parallel, approaches that implement the media application as a native application have also gained in importance: based on the Google Android OS[2], WMS services such as Google Android TV [Goo17c] and Amazon Video have also gained in importance. In these approaches, the MUI is a native client.

The differences in devices' performance of MUI execution are manifold, as these have diverse MSEEs, varying Operating Systems (OSs), and different MUI rendering engines. Expensive high-end devices fulfill every performance requirement and create high user service demands on QoS and system latency, which providers, in turn, are expected to fulfill across all platforms.

However, some devices are legacy devices released over the last decade with old Central Processing Unit (CPU) generations, insufficient memory, and low-resolution graphics rendering profiles.

Another group of devices is low-end Set-Top Boxes (STBs) and High-Definition Multimedia Interface (HDMI) dongles [A. 14]. These devices are available at a very low price, but the majority of such devices are also incapable of running regular MSEEs.

Besides, hardware upgrade cycles are much longer than the speed of software developments, so the difference between hardware device capabilities and service-driven developments is nearly impossible to overcome[3].

All things considered, providers should develop customized MUIs for each user device platform, which is a very fragmented device base, e.g. SmartTVs, Tablets, Smartphones, Desktops, etc. These are the most expensive and time-consuming types of development [C. 14] [EUR14].

In such a fragmented device market [D. 08], the integration of third-party applications presents another hurdle in the delivery of a superior user experience. In order to integrate these applications, providers are sometimes forced to provide dedicated application ecosystems that are not part of the providers' platforms. This results in an additional integration overhead when onboarding third-party services.

Often such integrations must deal with frequent application updates. It must also be ensured that such third-party integrations do not create any security vulnerabilities, as that could put the entire system at risk and cause significant damage to the platform.

Fragmentation also impacts the security standards of WMS platforms. Browsers have revealed multiple security vulnerabilities in recent years that could lead to malware installations or even attackers taking over control of a client device [Ath+15] [Ege+09] [Rei+09].

Different manipulations of high-value content by adversaries would then be possible. These might range from the personal use of content in an illegal way to the establishment of large-scale redistribution frameworks of illegally acquired media content [J. 16]. This results

[2] Android OS [Goo17b] has the biggest platform reach and the most significant amount of support in the software community.

[3] With advent of high-quality open source software frameworks and their growing support community, this difference might only increase in the future.

in providers losing revenue through users that register to and pay for such services without any knowledge of their illegal nature.

In summary, device fragmentation causes a multitude of operational issues in delivering media services to customers. These issues expose the gap between high user expectations for superior MUI and provider capabilities in providing consistent user experience across multiple devices. This gap is subsequently referred to as the *MUI Expectations-Capability Gap*.

1.2 Problem Statement

The previous section analyzed the current *MUI Expectations-Capability Gap* between user expectations for consistent MUI quality on multiple devices and the technical constraints of providers when delivering such services.

Furthermore, the constraints involved in developing a unified cross-platform MUI were analyzed to describe the challenges for the provider. This thesis addresses this gap by considering the fundamental aspects of the problem as stated below.

Problem 1 - **Device fragmentation:** in the context of device fragmentation, the technical requirements of WMS providers differ from those of their users: users demand a seamless and consistent service experience across all devices, such as e.g. Smartphones, Tablets, and SmartTVs. WMS providers, in turn, seek to serve all available device platforms without needing to adjust MUIs.

Within this constellation, device manufacturers are also pursuing their own interests and do not wish to align their technological preferences with those of their competitors. These preferences are their manufacturing secrets and unique selling points that enable market differentiation.

Despite all this, WMS services should be consistently accessible for users and still cross-deployable, without too many constraints for providers, on a multitude of device platforms. In this context, a service execution model that could bridge the *MUI Expectations-Capability Gap* is still missing.

Problem 2 - **Security threats of fragmented devices:** as outlined in the previous chapter, currently available MSEEs have proven to be vulnerable to security attacks. The number of security threats increases when considering the multiple device platforms and device types: each platform requires its security updates and patches, which have a high impact on the complexity of MUI developments.

Also, onboarding of third-party applications puts the streaming platform at risk, as they introduce additional threats. The ability of users to install apps on the device from app portals also creates vulnerabilities, as the portal might be running malicious apps as shown in [Zho+12].

Content piracy remains a significant threat for content providers, as media content has always been subject to extensive theft. Content providers have struggled to eliminate unauthorized access to premium content [J. 16].

For this reason, security is crucial for WMS services. Currently, there are no service execution models that reduce, by design, the risk of attacks on WMS platforms.

The next section defines more specific problems through the presentation of the scope of this work.

1.3 Scope

This section defines the focus of the research presented in this thesis. The primary goal is to design a service execution model that addresses the growing problems of a highly fragmented and insecure device market, identified above as Problems 1 and 2.

The service execution model that is developed in this thesis does not attempt to address the problems on the client-side but rather proposes a shift of the complete execution of MUI to a remote machine in the Cloud: the MUI is developed only once for the remote machine and then managed remotely out of the Cloud environment.

The proposed service execution model is presented in Figure 1.1 and has been named *Secure Remote Service Execution (SRSE)*. Combined with media delivered by the Media Server and rendered into one stream, the entire MUI is then provided to the client device.

Such a model addresses the security of WMS devices by design, as none of the local code is executed on the customer's device. The counterpart client that communicates with the server is the lightweight *SRSE Client* that is executed on a customer device. In this work, a locally run MUI is referred to as *Local MUI*, a remote MUI as *Cloud MUI*.

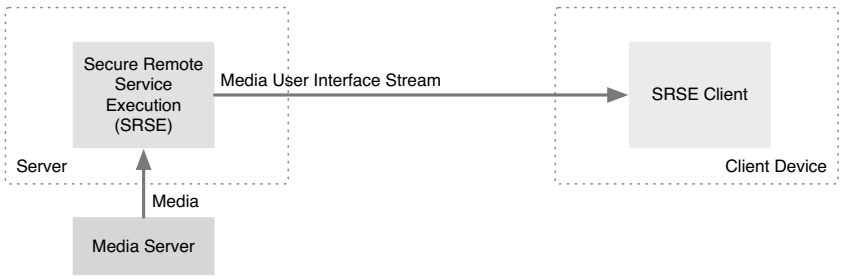


Fig. 1.1.: High-Level Representation of Secure Remote Service Execution (SRSE)

The introduction of the SRSE raises additional problems, as stated below:

Problem 3 - Security of SRSE: The security of the proposed model still needs to be analyzed. The SRSE Client executed on the client device is located on the user premises and establishes an interface with a user by processing his/her commands.

The security of the SRSE Client must be ensured to secure the data being sent to the SRSE, as the SRSE becomes a critical component serving many SRSE Clients. Any possible outages of SRSE should be eliminated.

Problem 4 - Bandwidth required by using the SRSE: The network load increases when moving to the SRSE model. Current network capacity might not be sufficient for a

transition to the SRSE, especially with regard to the broadcast of live events to a high number of users.

Problem 5 - **Execution of traditional technologies in SRSE:** The SRSE disrupts any conventional service currently supported by locally executed MSEEs. Herewith, both traditional browser- and TV-related technologies are affected[4]. As the local MSEEs are not available in the SRSE, the traditional execution of technologies is not supported.

Problem 6 - **SRSE Interfaces:** While browsers were evolving to MSEEs, standardization bodies completed a large amount of work on the definition of platform-related interfaces. These interfaces are required to enable a local browser to communicate with the platform resources of a device[5].

Analog to this, the SRSE model exposes missing communication interfaces between the SRSE and the device. A definition of interfaces is required to enable SRSE access to local resources of devices and the execution of traditional browser- and TV-related technologies.

Due to the support of browsers as MSEEs by various standardization bodies as discussed earlier, a Web browser has been selected by the author to analyze the SRSE model in a more concrete fashion. To address Problem 3, the so-called *Secure Cloud Browser (SCB)* architecture is designed and presented. Content protection mechanisms are considered as *security functions* to execute this model in a secured manner.

To resolve Problem 4, the SCB architecture is adapted to a limited network bandwidth capacity. This is achieved through two SRSE sub-approaches with respect to the delivery to the client:

(i) *Single Stream* approach: the SCB combines media and the MUI into one single media stream and delivers it to the client device;

(ii) *Double Stream* approach: the SCB renders the MUI only, thereby the media and MUI streams are delivered separately to the SCB Client that is responsible for putting the streams together and presenting them to the user in a combined form.

To solve Problems 5 and 6, a sub-set of traditional media technologies has been chosen for further examination. The choice of technologies has been derived from the author's contributions to W3C[6] and is considered in this work in order to prove the adaptability of SRSE to real-world deployments.

Browser-related technologies, developed by the World Wide Web Consortium (W3C) [Wor16], include (1) *streaming* - HTML5 [Htm] Media Source Extensions (MSE) [Col+14] and (2) *protected streaming* - HTML5 Encrypted Media Extensions (EME) [Dor+13]. This thesis focuses on *streaming* and *protected streaming* technologies as these are the foundation of every streaming service.

[4]The separation of technologies into TV- and browser-related does not highlight their current applicability, but rather identifies the origin of a technology. The browser-related technologies originate from browser developments on the Web; the TV-related emerged out of the digital TV domain.

[5]In the mobile domain the major part of the work has been completed by the World Wide Web Consortium (W3C) [Wor16]; in the TV domain by HbbTV and Open IPTV Forum (OIPF) [Ope16] bodies.

[6]The author has contributed to the W3C Cloud Browser Task Force [Clo17], which is part of W3C Web & TV Interest Group [W3C].

TV-related technologies comprise (3) *interactive service*[7] - support of HbbTV that enables HbbTV Application Signaling, i.e. HbbTV Red-Button signaling.

HbbTV technology is considered to be the most widespread TV-related technology in Europe. It is beyond the scope of this thesis to address how the HbbTV Red-Button will be executed by SRSE, in this thesis only HbbTV Red-Button signaling is discussed.

The architectures of adaptation to the SCB for each of these technologies are developed in the course of this thesis. They are referred to later as *Secure Streaming* for (1) MSE and (2) EME and *Secure Interactive Contents* for (3) HbbTV Red-Button signaling.

While adapting the chosen technologies to the SCB, known security vulnerabilities are considered and countermeasures deployed to make the designs secure. These vulnerabilities are stated further as Problems 5.1 and 5.2.

The missing interfaces are derived from a specification of the architectures. Specific components[8] are deployed in practical implementations according to their specifications. The feasibility of traditional technologies in SRSE prove that SRSE is a viable service execution model.

Secure Streaming (SES):

Different Adaptive Bitrate (ABR) streaming technologies and standards have emerged on the market, e.g. Apple HTTP Live Streaming (HLS) [App17c] or MPEG-Dynamic Adaptive Streaming over HTTP (DASH) [ISO12a].

In ABR streaming, a media file is divided into multiple segments that are encoded using different bitrates and resolutions. ABR streaming requires an ABR client that implements the ABR logic.

The logic, depending on a screen resolution and device network conditions, identifies and queries the required media segments at the server. This logic can be executed natively by the device platform, and can therefore be integrated with a Web browser over a plugin.

However, browser plugins have proven to be vulnerable to multiple attacks in the past [Bar+10] [Ban+10] [Kap+14]. Thus, to overcome the vulnerability of plugins and minimize the attack surface, the W3C defines MSE and EME browser interfaces that enable the JavaScript to call native libraries.

The W3C MSE interface allows an HTML5 application executed in a Web browser to handle the ABR logic and implement a cross-browser interoperable ABR client. MSE, therefore, enables ARB streaming in HTML5.

W3C EME is an interface that allows the playback of encrypted video in HTML5. EME is a JavaScript Application Programming Interface (API) that allows an HTML5 application to access the Content Decryption Module (CDM). CDM includes a Digital Rights Management (DRM) implementation for playback of encrypted media contents.

[7] Other browser-related technologies include e.g. multi-device interactions and TV-related technologies comprise e.g. TV-tuner functionality or access to the Electronic Program Guide (EPG) metadata.

[8] The client-side components are within the scope of implementations in this work. The server-side components are beyond the scope of implementations and are chosen from solutions that fulfill the requirements identified in Chapter 3.

Problem 5.1 - Insecurity of EME: In the context of EME, a DRM scheme and a graphics library are closely integrated with each other [Mic17d] [A. 17] [Ggl]. This poses vulnerabilities in the software-based CDMs integrated with no hardware assistance, as browsers were initially designed for unencrypted media playback.

As proven in [MK14], graphics libraries are vulnerable, as they run in user mode and are often based on open source libraries, e.g. FFmpeg [FFm17]. This increases security threats to EME, making content stealing attacks possible [K. 16].

Moreover, in February 2015 the Hollywood Research Laboratory Movielabs [Mot17] issued a specification for high-value content [Mot15], where DRM execution in a Trusted Execution Environment (TEE) is required.

TEE enables hardware-assisted decryption of the content for high-value content. Some of the CDM implementations available on the market are already integrated with TEE. However, these solutions are proprietary and closed-source.

Owing to the nature of the local execution of MSE/EME interfaces, they are extended in this thesis for execution within the SCB. To address Problem 5.1 and enable secure EME integration with the SCB, the EME interface is securely designed with an isolation of decryption keys in TEE.

In contrast to the hardware-assisted EME solutions that have recently appeared on the market [Mic17b] [Ggl], this thesis focuses on the implementation of open source technologies for TEE.

Secure Interactive Contents (SIC):

This thesis focuses on the *interactive service* enabled by the European standard HbbTV. HbbTV introduced Web content to traditional broadcast television, embedding HTML pages into broadcast video streams. It also gave TV sets the ability to render these pages and present the content to users while enabling interactive applications.

In HbbTV, signaling of applications is processed by special libraries and an HbbTV-enabled browser on the device. The shift of MUI execution to a remote machine directly affects HbbTV as signaling is traditionally processed on the device.

Problem 5.2 - Insecurity of HbbTV signaling: In May 2014 researchers theoretically displayed the insecurity of HbbTV [OK15]. The lack of authentication for broadcast stations in devices, i.e. receivers in Digital Video Broadcasting (DVB) [ETS10b], enabled tampering with the broadcast signal and a further use of HbbTV to perform attacks.

These attacks could be performed by DVB receivers on pages hosted on the Web, as well as the other way around, exploiting bugs in DVB receivers, e.g. Smart TVs [MK14]. The last attack was executed in 2015 [Mic15].

These attacks were made possible due to a significant finding made earlier [Ghi+13] [GT14] that shows an unusual behavior in HbbTV applications: the HbbTV applications could be pre-loaded and executed before a user actively triggers them by pressing a red button on a remote control. Therefore, these applications could be launched by attackers remotely without any knowledge of the user.

To enable HbbTV functionality in the SCB platform, new architecture and interfaces are defined in this thesis. Addressing Problem 5.2 of HbbTV insecurity, HbbTV is designed to be secure by integrating HbbTV Red-button signaling with TEE.

In summary, Figure 1.2 depicts the research scope of the thesis and the technologies considered for analysis. SCB architecture is designed as a special case of SRSE. The SES and the SIC services are proposed by adapting the traditional services to the SCB. Finally, the missing interfaces for the services are developed.

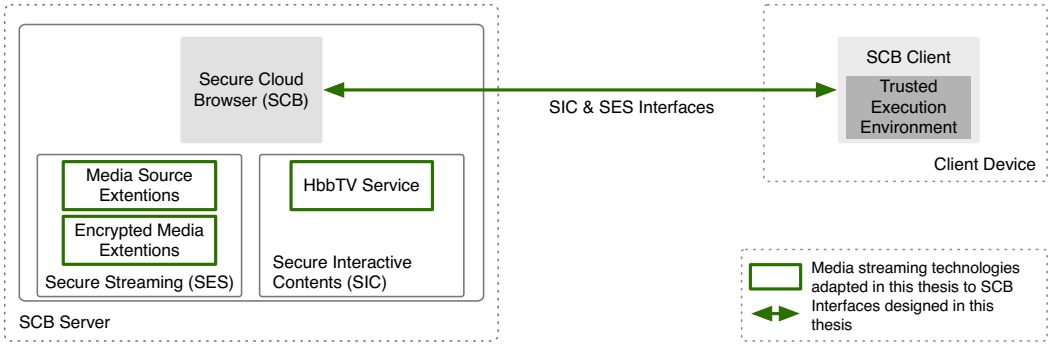


Fig. 1.2.: The scope of the research presented in this thesis: Secure Cloud Browser architecture, its services and interfaces

1.4 Research Questions

This section summarizes the scope of the thesis and narrows it by defining the *Research Questions*: the Research Questions will be addressed throughout the thesis and finally discussed in Chapter 7 to summarize this thesis' achievements. This thesis addresses the Research Questions stated below.

Research Question 1: How can the shift of local MSEEs into the Cloud be executed with a design for the SRSE that preserves the security of high-value content?
Research Question 2: What impact does the SRSE model have on current network infrastructures?
Research Question 3: How can *streaming* be realized for the SRSE platform? Which new interfaces must be defined for this realization?
Research Question 4: How can *protected streaming* be realized for the SRSE platform? Which new interfaces must be defined for this realization? Is it possible to enhance the security of *protected streaming* through the use of a TEE that isolates sensitive data? What would the overhead of such advanced security be?
Research Question 5: How can *interactive service* be enabled for SRSE? Which interfaces should be defined for remote execution? Is it possible to protect such a system through TEE against the attacks defined earlier? What is the additional execution time for such security measures?

Therefore, the main contributions of this thesis are:

Contribution 1: Architecture and specification of the SRSE platform with two different approaches for MUI delivery to the device: *Single Stream* and *Double Stream* approaches, where the second approach addresses the need to save bandwidth.

Contribution 2: SES architecture for remote processing, definition of new SES interfaces and design of advanced security through the integration of EME with TEE, including an analysis of the overhead caused by the advanced security.
Contribution 3: Design of SIC architecture and corresponding interfaces for remote processing, protection of HbbTV Red-Button signaling through TEE and analysis of the additional execution time required for such protection.

1.5 Research Methodology & Outline

The research in this thesis follows the methodology used for the design of secure systems presented in [McD91]. The methodology is built upon following design stages: (1) analysis of state-of-the-art technologies; (2) definition of requirements for the desired architecture; (3) the design of the overall architecture and its specification; (4) implementation of designed architecture, according to the specification, in order to prove the feasibility of the proposed design; (5) evaluation of the research, its challenges, and open questions.

Governments and different Standard Developments Organizations (SDOs), e.g. W3C, have also adopted this methodology for the establishment of standards. The viability and research applicability of this methodology has therefore been proven in various research activities subsequently published as standards.

The structure of the thesis as defined below reflects the methodology model. Chapter 2 describes and analyzes in-depth the MWS and all related technologies that are of importance for this thesis. First, the MSEEs are examined. Afterward, the following services and corresponding security-related issues are analyzed: *streaming*, *protected streaming* and *interactive service*. The chapter is then concluded with an overview of related research.

Chapter 3 identifies and analyzes the requirements for the SRSE and the services SES and SIC that are adapted for remote execution in the SRSE. Chapter 4 defines the functional architecture for SRSE and services, fulfilling the requirements identified earlier. It also provides a specification for the functional architecture and defines the required interfaces.

Chapter 5 implements the client-side components of the SRSE and the SES and SIC services. It also provides an insight into the implementation details.

Chapter 6 analyzes the architecture and related approaches presented in this thesis. This chapter also presents the measurements performed and evaluates the results received. Chapter 7 draws conclusions from the results, returns to the Research Questions posed in Chapter 1.4 and discusses directions for possible future research.

1.6 Author's Publications

This section presents publications of the author that are within the scope of the thesis. The thesis has taken them as its foundation and developed them further:

P1 A. Mikityuk, J. P. Seifert, and O. Friedrich. „The Virtual Set-Top Box: On the shift of IPTV Service Execution, Service and UI Composition into the Cloud“. In: *2013 17th International Conference on Intelligence in Next Generation Networks (ICIN)*. 2013, pp. 1–8

P2 A. Mikityuk, O. Friedrich, and R. Nikutta. „HbbTV Goes Cloud: Decoupling Application Signaling and Application Execution in Hybrid TV“. In: *Proceedings of the ACM International Conference on Interactive Experiences for TV and Online Video*. TVX '15. Brussels, Belgium: ACM, 2015, pp. 191–196

P3 A. Mikityuk, S. Pham, S. Kaiser, O. Friedrich, and S. Arbanowski. „Content Protection in HTML5 TV Platforms: Towards Browser-agnostic DRM and Cloud UI Environments“. In: *Proceedings of the 5th International Workshop on Trustworthy Embedded Devices*. TrustED '15. Denver, Colorado, USA: ACM, 2015, pp. 43–52

P4 D. Livshits, A. Mikityuk, S. Pham, and A. Shabtai. „Towards Security of Native DRM Execution in HTML5“. In: *2015 IEEE International Symposium on Multimedia (ISM)*. 2015, pp. 411–416

P5 A. Mikityuk, M. Platschek, and O. Friedrich. „On Virtualization of Red-Button Signaling in Hybrid TV“. In: *2015 IEEE International Symposium on Multimedia (ISM)*. 2015, pp. 435–440

The author was the leading contributor to all publications except for Paper 4. The publications that are related to the research topic but do not comprise the foundation of the thesis are analyzed in Annex A.

Paper 1 introduces an overall approach to the concept of the SRSE. The work focuses on the creation and delivery of media services in IP Television (IPTV). A concept for the so-called virtual STB is presented in this paper, in which most of the MSEE - e.g. the Web browser - is shifted to a Cloud infrastructure. Chapters 2.1.1 and 4.1 in this thesis represent and extend the contributions of this paper.

Paper 2 presents an architecture that enables the shift of the *interactive service* HbbTV into the Cloud. This is based on the decoupling of HbbTV Red-Button signaling and HbbTV application execution on the device. The shift is executed by defining new interfaces for service-to-cloud and cloud-to-device.

Paper 5 continues with this topic and identifies a challenge caused by device fragmentation on the HbbTV market. The fragmentation arises due to versioning of the HbbTV standard, various hardware capabilities of HbbTV devices and the lack of HbbTV standard support on millions of devices. The challenge of fragmentation is addressed in this paper with a so-called *Cloud-enabled HbbTV* concept.

This concept is based on the virtualization of HbbTV Red-Button signaling. The Red-button signaling is executed and handled within the Cloud. The contributions of these papers are presented and extended in Chapters 2.3 and 4.3.

The integration of DRM through a W3C EME interface into a Web browser resulted in a disruption of how DRM systems work. Desktop browsers like Google Chrome [Goo17f], Mozilla Firefox [Moz17b] and Microsoft Edge [Mic17a] already support the EME standard, but are limited to one single DRM.

As a result, browser manufacturers are dictating the DRM system that should be used [Pha+16]. Paper 3 presents an open architecture for the decoupling of DRM from the Web browser. This enables the exchange of DRM schemes and therefore opens up the possibility of integrating multiple DRMs. This paper also defines a target architecture for EME adaptation in the SRSE.

Paper 4 deals with an open, secure and flexible architecture for integrating EME with TEE. This provides security hardening for the playback of DRM-protected content without any need for a dedicated secure processor. This paper is not focused on specific TEE approaches

but rather considers the possibility that any TEE could potentially be integrated with CDM through the EME interface.

This work introduces the methods for integrating CDM with Intel Software Guard Extensions (SGX) [Int17] and ARM TrustZone [ARM17a] TEE technologies. The contributions of Papers 3 and 4 are presented and extended in Chapters 2.2 and 4.2.

State-of-the-Art

2

This chapter is devoted to current State-of-the-Art (SoA) technologies and related works in the field of service execution for Web Media Streaming (WMS). First, the most relevant for this thesis Web media streaming technologies will be analyzed. This analysis will include a detailed explanation of the technologies and commonly used definitions in the Web streaming domain. Building on this analysis, an extended study of related works presented in academia will follow.

This chapter is structured as follows. It begins with a description of service execution for Web media in Chapter 2.1. Here, the service execution principles are demonstrated through a detailed analysis of execution environments currently available for WMS, the so-called Media Service Execution Environments (MSEEs). Due to technology standardization and full acceptance in deployments, the browser-based MSEE is within the scope of this research.

Further, a study of technologies is done with an emphasis on the browser and how these technologies could be integrated into the browser landscape. For this work, three media services have been selected to prove their feasibility in the Secure Cloud Browser (SCB) landscape: streaming, protected streaming and interactive services.

Chapter 2.2 studies the relevant streaming technologies introduced over the last decade on the Web. Another important aspect analyzed here is the protection of streaming through the introduction of content protection technologies. For the advanced security of the currently available protection mechanisms, a detailed study of Trusted Execution Environments (TEEs) is presented here.

Interactive services that bring interactive applications to Smart TVs will be analyzed in Chapter 2.3. After the study of relevant technologies has been presented, research regarding relevant concepts in academia is introduced in Chapter 2.4.

2.1 Media Service Execution

Service execution is a rather broad term, and its definition depends on a particular field of application. Here, service execution is studied as a platform for the consumption of Internet media, i.e. MSEE.

Chapter 2.1.1 provides an introduction to this topic and describes the landscape of existing MSEEs. Chapter 2.1.2 gives an overview of the browser-based MSEEs, as these are within the scope of works presented in this thesis. Chapter 2.1.3 studies the insecurity of Web browser technologies. Annex B.1 briefly introduces the OS-based MSEEs.

2.1.1 Media Service Execution Environments

An important inflection point on the way of the Internet development towards a universal service platform was the release of the Mosaic Browser[1] [Nat]. It was the first easy to use and free tool for consumption of Internet services that increased the demand for Internet services. The range of Internet services on offer has grown to meet the customer's needs.

Now that the Internet has become a platform for media, the tools for its consumption have also evolved. These tools are MSEEs that enable execution of media-related services[2]. To run the services, MSEEs comprise runtime environments, libraries, frameworks and APIs that are specific to end-devices' hardware [Fri11]. The MSEEs facilitate the integration of services on end-devices and partially abstract the integration from platform hardware.

The choice of the MSEE depends on the way the media service application is implemented[3]. If it is implemented as a Web application, a Web runtime environment is required, and the MSEE would be the browser. If it is implemented as a native application, then the MSEE is an Operating System (OS), e.g. Android OS [Goo17b].

The advantages of the Web-based approach are the Web standards provided by W3C [Wor16] that provide a high level of technology acceptability. The standards enable interoperability and a broad reach for services on multiple platforms through standard technologies.

Until recently, the performance of native approaches has been outpacing browser-based apps. However, with the advent of a hybrid approach, browser technology might soon become the first choice for MSEEs in media services.

The hybrid approach enables the wrapping of browser-based applications into containers that have access to the native APIs of end-devices. Hybrid applications require Web runtime environments and are developed with Cascading Style Sheets (CSS) [Css], HTML and JavaScript (JS) [Js2] technologies.

To get access to platform hardware, such frameworks as Manifold.JS [Man17], Apache Cordova [Apa17], Microsoft Xamarin [Mic17h], etc. are used. One difference of these applications when compared to native ones is that the MUI rendering is done via WebViews[4] and not by native platform frameworks. They also differ from Web apps in that they have access to native platform APIs, which is not the case in the pure Web approach.

In all these approaches, Local MUI, i.e. the MSEE and the media services, is executed locally on the end-device. The Local MUI model is a data-driven model, where the static code is executed on the end-device, and the data is supplied to the code upon request.

The following chapters will discuss in detail the Web browser as MSEE. Through the latest standards, such as W3C MSE and EME, the browser evolves to an interoperable and ubiquitous MSEE for Web media consumption.

In addition, the WAVE [Con16b] initiative aims to enable the next inflection point on the Internet for media distribution services. WAVE is doing so by putting together major Web

[1] The Mosaic Browser was released at the National Center for Supercomputing Applications (NCSA) [HH15] [P. 13] in 1993.
[2] Such services cover standard media use cases as Live TV, Video on Demand (VoD) or Program Guides, and also the now-emerging Augmented Reality, Virtual Reality, and 360° Media Streaming services.
[3] Possible application implementation approaches are listed in Annex B.3.2.3.
[4] WebView is a native OS component provided for the loading and representation of Web content.

and streaming standards and enabling a universal player tool for the consumption of media services.

With such a tool, the advent of the so-called Tier3 media distribution providers would be possible. In Tier3 distribution, presented in detail in Chapter 2.2.2.3, not only big corporations but also small clubs or individuals would have the chance to become an MWS provider quickly and easily.

2.1.2 Browser-based Approach

Browser developments have come a long way since the introduction of the first browsers that were simple viewers of HTML content. Now browsers are rich and universal client ecosystems for Web streaming media. They enable CSS and (X)HTML rendering of Web pages and execute their JS logic.

Beginning from HTML5, the browsers can also natively play back major media formats through the `HTMLMediaElement` interface. The media content can be easily embedded into a Web page through HTML5 `<video>` and `<audio>` tags. Previously, media playback was only possible through plug-in systems, e.g. the Adobe Flash Player [Ado17c] or Microsoft Silverlight [Mic17e], with `<object>/<embed>`.

The MSE and EME interfaces, studied in detail in Chapters 2.2.1.2 and 2.2.2.1 respectively, extend the `HTMLMediaElement` and enable native playback of DRM-protected Adaptive Bitrate (ABR) content. All these features are currently supported on the main desktop browsers that include Google Chrome, Mozilla Firefox, Microsoft Edge and Apple Safari [App17e].

In the mobile and embedded domain, a native-based approach to application development dominates. However, this might change with an advent of hybrid applications. In this case, the features of desktop browsers will also be adopted by mobile and embedded browsers.

With regard to cross-platform compatibility, the real-world implementations of different browsers remain inconsistent, despite the W3C standards. The browsers might interpret CSS and HTML code differently, and they might also be different in a way they implement JS [Rad15]. Also there are differences in the media formats and DRM systems that the browsers support[5].

There are JS frameworks like jQuery [The17a], AngularJS [Goo17d], etc. that aim to address the challenges in Web application development on the way towards possibly addressing all devices with the deployment of one UI. These frameworks can significantly reduce development efforts. However, they do not eliminate challenges concerning browser versioning and issues with different implementations of Web technologies on various platforms [Rad15].

In summary, Web browsers are supported on multiple platforms, have a big developer community and a variety of frameworks already available. Through the hybrid approach, the browsers might become the dominant technology for Web media MSEEs on the market.

[5] Chapters 2.2.1, 2.2.2.1 and Paper [Pha+16] takes a deep dive into the media format diversity and DRM inconsistency across browsers.

2.1.2.1. High-level Architecture

Web browsers retrieve data from Web resources by downloading it to a user's machine, and display the data by rendering it. Subsequently, through navigation on the displayed content, they access further data on Web resources that are required by the user.

To retrieve the data, the user would insert the so-called Uniform Resource Locator (URL) into the address field of the browser. The URL points to the required Web resource. Once the resource data has been retrieved, the browser displays it to the user. The displayed content is the UI.

The high-level architecture of a browser is shown in Figure 2.1. Underneath the UI that displays the content, lies the core part that is the browser engine. Different browsers use different browser engines, e.g. EdgeHTML [K. 17] for Microsoft Edge, WebKit [App17f] for Apple Safari, Blink [Goo17e][6] for Google Chrome and Gecko [Moz17a] for Mozilla Firefox.

Browser engines embed so-called rendering engines that are responsible for displaying data on monitors. Another important part that enables the interactivity of the Web page is the JS engine. For Webkit, the built-in JS engine is the JavaScriptCore [App17d] and is used by Apple Safari and a mobile version of Safari. Google uses its implementation of the JS engine which is V8 [Goo17i].

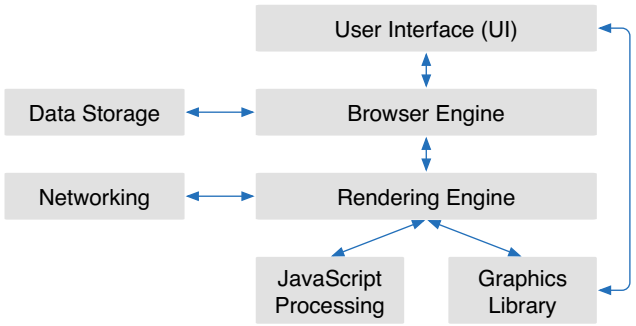


Fig. 2.1.: Web Browser High-Level Architecture

The rendering engine[7] implements Web technologies in order to render the following Web content: HTML, CSS, Document Object Model (DOM) [Dom], Scalable Vector Graphics (SVG) [Svg].

It is important to mention that browsers implement the concept of *platform abstractions*, so they can be executed on multiple platforms. *Platform abstractions* mean that if two browsers use the same rendering engine implementation, the way they output the data to the device display is different as different device APIs are used, e.g. networking, geolocation, graphics APIs, etc. Therefore, the UI is different on different platforms, as these abstractions are based on various APIs.

[6] Initially, Blink was forked on GitHub [Git17] from WebKit.
[7] For example, the rendering engine of Apple WebKit is the WebCore.

2.1.2.2. Functioning

When the user requests the URL, the first function that is responsible for retrieving or fetching the data from Web resources is the *loader*. Afterward, the *parser* goes through the retrieved data and parses the data to perform its syntactic analysis. Parsers for HTML and CSS data are called *HTML Parser* and *CSS Parser*. To build the so-called layout of the UI, the *HTML Parser* constructs the *DOM Tree*[8] and the *CSS Parser* constructs the style rules. These are then both combined to build up the *Render Tree*.

Then, the *painting* procedure begins. In this step the objects of the *Render Tree* are painted to the UI. The graphic commands for *painting* are sent to the so-called *Graphics Context* abstraction. Graphics Context abstraction implements different Graphics Libraries on different platforms, e.g. CoreGraphics [App17a] used by Apple and Skia [Ski17] used by Google and Mozilla. After the *layout* is painted, the Web data is presented to the device's display.

2.1.3 Web Browser Insecurity

Despite continuous efforts of vendors to improve the security of Web browsers, they remain targets for a variety of security attacks [Tsa+15] [Vir+15] [Vir+14] [Tsa+14]. Web browsers are attractive targets for adversaries as they are involved in high-value content consumption, payment transactions, operations with user data and other critical processing of sensitive data.

Since the introduction of a Web browser, a variety of vulnerabilities has been revealed in academia and real deployments. This chapter exemplarily shows some of the known attacks.

To increase the efficiency of Web browsers, the vendors started to integrate JS interpreters with compilers into so-called Just-In-Time (JIT) engines. They are deployed across all browsers and are part of JS engines.

Athanasakis et al. in [Ath+15] prove that despite latest security preventions, the JIT engines remain vulnerable to exploits with dynamically generated shellcode, i.e. gadgets, injected during JIT compilation.

Two JIT engines do not withstand the attacks with dynamic Return-Oriented Programming (ROP) payload construction presented in the paper: both the open source JIT engine from Mozilla Firefox and the proprietary JIT engine in Microsoft Internet Explorer (IE). Athanasakis et al. demonstrate proof-of-concept attacks and manage to run shellcode in both browsers successfully. They also bypass a variety of defenses against JIT exploitations in IE.

Kapravelos et al. analyze malicious extensions that affect the Chrome browser in their work in [Kap+14]. The authors present their system Hulk, which performs a dynamic analysis of malicious behavior in browser extensions. Hulk creates dynamic web pages that continually adapt to the behavior expected by extensions. In such a way, Hulk triggers the designed behavior of the extension and analyzes whether it might be ranked as malicious or not.

[8] The DOM Tree is the logical structure of the DOM.

The authors analyzed 48K extensions available on the Chrome Web store running each with 1M URLs. Out of 48K extensions, their system ranked 130 as malicious and 4,712 as suspicious. Together they have 5.5M browser installations.

In their work, the authors also show concrete examples of malicious extensions, e.g. WhasApp that resembles the popular WhatsApp. One of the targets of this extension is Facebook, where the extension injects URLs to spread itself further and to monetize users through product advertisements.

2.2 Web Media Streaming and its Protection

Media streaming is evolving at the speed of the Web and video is becoming the most important traffic type. Chapter 2.2.1 describes the different Web streaming technologies currently available. The protection of the media through the DRM system is analyzed in Chapter 2.2.2. Advanced security architectures, TEEs, are examined in Chapter 2.2.3.

2.2.1 Media Streaming on the Web

Today, media streaming on the Web is represented by three technologies: HTTP progressive download, real-time media streaming protocols and adaptive HTTP streaming protocols. Chapter 2.2.1.1 describes the adaptive HTTP streaming protocols, which are gaining in relevance for Web-delivered media tremendously.

To consume Web streaming on the Web, in most cases the browser is the most deployed and utilized media consumption technology. Chapter 2.2.1.2 examines browser support of ABR streaming.

An overview of HTTP progressive download and real-time media streaming protocols is provided in Appendices B.2.1.1 and B.2.1.2 respectively. Appendix B.2.2 will present a summary and comparison of streaming technologies with each other.

2.2.1.1. Adaptive HTTP Streaming Protocols

ABR addresses the drawbacks of progressive download and real-time streaming protocols, described in Appendices B.2.1.1 and B.2.1.2. ABR combines the advantages of content delivery over HTTP, which allows for serving large numbers of clients, with efficient segment-based content distribution. Additionally, as the name suggests, this streaming technology allows for adaptations of the stream to different network conditions, also tolerating bad ones. The ABR protocols include proprietary protocols like HLS and Adobe HTTP Dynamic Streaming (HDS) [Ado17b] and an open standard protocol MPEG-DASH.

HTTP protocols are stateless. Therefore, a server does not have to maintain persistent connections to clients. The server sends data packets upon client requests and closes connections right after the data has been sent over. Due to this stateless connection, HTTP Web servers can serve many more clients than real-time protocol's servers.

In ABR, the media file is divided into small segments. These segments can vary in their length, but they mostly are 5 to 10 seconds. The segments are encoded at multiple bitrates. The amount of available different bitrates is valid for all segments and depends on the

encoding settings set by media content distributors. Usually, the available bitrates cover the worst and the best available network conditions and values in between. With this, end-users are enabled with a seamless viewing experience even if the network quality is bad.

These multiple representations of every segment of the media file are available on the server-side. The server-side might include multiple server locations that are part of one Content Delivery Network (CDN). After the media file is segmented and the segments are encoded, a so-called manifest description is established and hosted together with the media data. The manifest is the first file that is sent to the client, and it includes a description of the segments' ordering and where the corresponding segments are located.

Based on that, the client requests the segments using the HTTP GET method. The client also has full control of the media playback. Based on estimations of available network bandwidth or processor load it might request higher or lower bitrates of the content segments. This enables media quality adaptations on the fly and makes the protocol highly bandwidth efficient.

Being the core traffic of the Web, HTTP traffic is currently highly optimized through the various caching algorithm. Each SoA CDN enables a highly efficient HTTP delivery through distributed caching systems. Therefore, by using this streaming technology, these protocols have been rapidly adopted by the majority of streaming providers.

This rapid adaptation speed has demanded one standard solution since no universal standard existed before. Three main protocols were available on the market: HLS, HDS and HTTP Microsoft Smooth Streaming (HSS) [Mic17f]. The DASH standard has since harmonized the landscape and become the most adopted standard on the Web. However, HLS is still required for mobile Apple iOS platforms and MacOS, when encrypted content is included.

Dynamic Adaptive Streaming over HTTP (DASH) The ISO/IEC 23009 [ISO12a] specification has harmonized ABR protocols and presented MPEG-DASH. DASH [Sto11] is an open international industry standard for ABR. As different browsers support different video codec technologies [dev16], this becomes a constraint when adapting a streaming technology.

Unlike the proprietary ABR protocols, DASH is video codec agnostic. DASH supports ISO Base Media File Format (ISOBMFF) [ISO15] containers that consist of non-multiplexed video and audio tracks. ISOBMFF is a modular file format that consists of data structures, or so-called boxes, containing initialization data, metadata, and media segments. These segments are encoded in various bitrates that are then requested by the client.

2.2.1.2. Media Streaming Support in Browser

Before HTML5 was introduced, HTML did not have any native support for media playback. To address this issue, proprietary media plugins for browsers such as Adobe Flash and Microsoft Silverlight have been developed. Through the adaptation of Flash to main platforms, it was seen as an interoperable media playback technique. However, multiple security vulnerabilities, e.g. cross-site scripting, cross-site flashing [S. 07b] [S. 07a] [fuk07] and parameter injection attacks [Bar08] [S. 08], were a constant threat for end-users.

Adversaries were continuously exploiting these security vulnerabilities to install malware onto end-users' machines. Unceasing security attacks on Flash led browser manufacturers

to decide to introduce native video playback in HTML5. For this reason, Apple has stopped Flash support on their mobile devices [S. 10].

This decision was adopted across the industry, and Adobe finally announced the end of Flash support on mobiles [Ado15]. Currently, desktop browsers as well, such as Google Chrome, Microsoft Edge, and Apple Safari have stopped to support Flash, using the native media playback of HTML5 instead.

HTML5 `HTMLMediaElement` interface enables native video and audio playback in HTML through corresponding `HTMLVideoElement` and `HTMLAudioElement` interfaces. The HTML `<video>` and `<audio>` elements enable a standard way to embed media into Web pages. Today, the primary platform for the playback of ABR streaming on the Web is a browser. To address ABR, as described in Chapter 2.2.1.1, the MSE specification was established by W3C.

MSE extends the `HTMLMediaElement` to support ABR in HTML and provides a standard JS interface for playing ABR content in the browser. Using this standard Web API, an HTML5 media application can provide a JS ABR client that will be interoperable with every browser platform. The client would handle the ABR logic for media stream reconstruction, bitrate adaptation, etc., therefore providing the application with full control over media streaming. One example of such an interoperable ABR playback client for DASH is an open source dash.js player [DAS16b], provided by DASH Industry Forum (DASH-IF) [DAS16a].

Concerning MSE, the JS player could be downloaded as part of the application. While embedding ABR into HTML, the media source attribute `src` does not point to a media file but rather to a manifest data `manifest.mpd`. In the manifest data, the media description will be provided to a client, and the media played back in e.g. `dash.js` player.

Media Source Extensions (MSE)

With a progressive download, the media file was saved on a hard drive of an end-device, and the HTML5 `<video>` element played it back through the graphics library of the platform. In ABR streaming, media segments are downloaded and appended to a buffer. With MSE, an application reads the data from a buffer and MSE enables the buffer as a source attribute `src`.

Instead of a file location or its URL, the source of the video `src` is the MSE `MediaSource` object. Therefore, this object, being a virtual URL for the HTML `<video>` element, acts as a buffer. The `SourceBuffer` objects are filled with media segments and appended to the `MediaSource` object.

The `<addSourceBuffer>` method appends the buffers to the `MediaSource` object. The data that is appended at this stage is a reconstructed media data that can be played back by the HTML5 `<video>` element. If the connection condition changes and the appended buffer cannot be played out, the `<removeSourceBuffer>` method removes the `SourceBuffer` from the `MediaSource`. The same mechanism is also at play in the media tracks of `SourceBuffer` objects.

Within a `SourceBuffer` object, multiple track buffers are filled with video and audio track data. These track buffers are appended to the `SourceBuffer` object, utilizing the `<appendBuffer>` method. Figure 2.2 demonstrates the object model of the MSE interface.

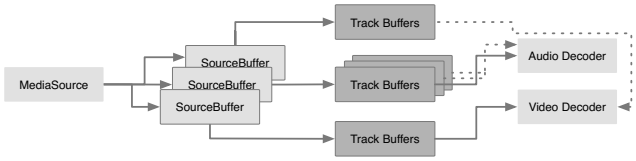


Fig. 2.2.: Object Model of the W3C Media Source Extensions (MSE) Interface (adapted from [Col+14])

2.2.2 Protecting Media Streaming on the Web

> *Digital files cannot be made uncopyable, any more than water can be made not wet [B. 01].*
>
> **— Bruce Schneier**
> (American cryptographer, computer security, and privacy specialist)

To protect valuable content while distributing it on a network, content protection mechanisms like DRM have been developed over the past decades.

Chapter 2.2.2.1 presents an integration technique for DRM in browser environments. Chapter 2.2.2.2 analyzes the limitations of current EME deployments and their insecurity.

Chapter 2.2.2.3 presents a summary of the overview of presented technologies, discusses their ethics and gives an overview of their future. In Appendix B.3.1 Conditional Access (CA) is studied. This is followed by a detailed analysis of DRM in Appendix B.3.2.

2.2.2.1. Web DRM: DRM Support in Browser

In 2012, W3C started to work on EME[9] extensions to the HTML5 standard. This new standard has been specified to overcome the need for proprietary plugins and technologies for protected content playback in browser environments. As stated previously in Chapter 2.2.1.2, MSE are extensions to the `HTMLMediaElement` that enable ABR media in HTML5. EME are also extensions to the `HTMLMediaElement` that enable the playback of encrypted media in HTML5.

The EME specification extends `HTMLMediaElement` and provides a JS API to control the playback of protected content that is decrypted by the so-called Content Decryption Module (CDM). This API allows a Web application to access the CDM.

This standard introduces two entities: the CDM component that is part of Web browser and EME JS interface exposed for Web application allowing communication with CDM. The Web application controls the license/key exchange.

The primary responsibilities of CDM are license processing, business rules validation and decryption of the content. CDM can be an independent module that implements all the required DRM functionality or on the contrary, can use native DRM services depending on the CDM provider.

[9] Netflix, Google and Microsoft have pushed the work on EME with the goal to make DRM a common browser feature.

CDM includes DRM implementation from the browser or the platform that plays back encrypted media contents. The CDM is loaded by the browser and provided by a DRM provider.

EME architecture allows browsers to contain multiple CDM modules that are provided by different DRM vendors. However, as previously mentioned, at present modern browsers are only capable of supporting a single CDM, if at all, e.g., Microsoft Edge uses Microsoft PlayReady DRM [Mic17c] and Google Chrome uses Google Widevine DRM [Goo17j].

The EME specification does not define the DRM systems, instead it defines an API that might be used to discover and interact with such systems. The implementation of DRM is not required for compliance with this specification. Only the implementation of the Clear Key system as a common baseline is required.

The Clear Key system utilizes AES-128 bit keys that are sent in cleartext over the network and therefore exposed to applications. The EME flow is explained below with regard to the MPEG-DASH streaming format and depicted in Figure 2.3. However, any streaming technology can be addressed by EME.

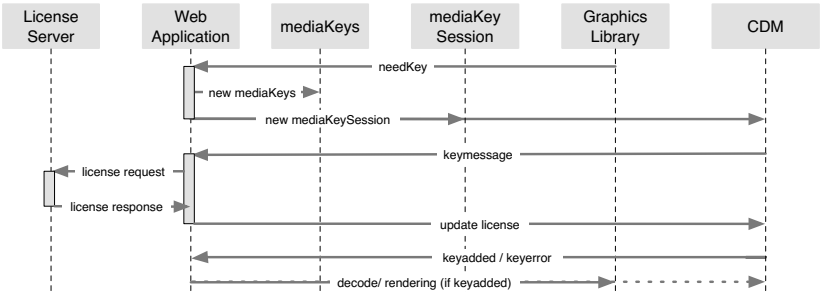


Fig. 2.3.: Message Flow of the W3C Encrypted Media Extensions (EME) Interface

In the EME context, the DRM-specific metadata is provided within the PSSH boxes of the MPEG-DASH media stream. These boxes contain the identifier `DRM Universally Unique Identifier (UUID)` of a DRM System and a key identifier of this particular DRM System. The following steps are required for the playback of encrypted content.

After the Graphics Library has parsed the media data packet and identified that the media content is encrypted, the Graphics Library generates a `needKey` event with DRM-specific metadata taken from the stream. The `needKey` event is received by the Web Application.

The Web Application creates `mediaKeys` and `mediaKeySession` objects, and triggers the CDM to generate the key message. `mediaKeySession` is an object that is responsible for the interaction between the Web application and CDM for the current playback session.

The initialization data `initData` information for the `mediaKeySession` is taken from the `needKey` event. The initial request to the CDM includes `initData`.

`initData` is a block of data containing information about the key that is needed to decrypt the stream. This information uniquely identifies the decryption key and is required to generate a license request. This data may come from the media files, a manifest file, such as the Media Presentation Description (MPD) file when using MPEG-DASH, or application-specific methods.

After the CDM has triggered the key message to acquire the DRM license from the DRM key server, the license is sent to the client. The `mediaKeySession` is then updated with the license information received. The decryption key is extracted from the license and content is decrypted after the business rules are verified.

After CDM decrypts content, there are two options: the content is decoded and rendered within CDM or is sent as plaintext to the Graphics Library of the platform for decoding and rendering, depending on CDM implementation.

2.2.2.2. Constraints of Web DRM and its Insecurity

EME was proposed in 2012. One of the goals of EME was to achieve flexibility for applications and content providers. However, EME specification is a pure Web browser technology. EME specifies only the API for applications running within the Web browser to interact with the CDM. The DRM decryption itself takes place within the CDM.

The CDM component is not defined and standardized in the EME specification. The CDM architecture is left completely open, with the following options: CDM is a part of the Web browser and implements the DRM scheme in a sandbox, CDM is a part of the device platform and is executed in an obfuscated software or CDM is a part of the device platform and is executed in a hardware-assisted manner.

The CDM functionality can be one of the following: CDM itself decrypts and decodes the media or CDM decrypts the media and passes it over to the graphics library. These options are based on the assumption that the connection between the CDM and the graphics library is considered to be a trusted channel in the EME specification.

The lack of CDM specification results in an interoperability issue among CDMs: most of the browser vendors are embedding their own platform-specific and proprietary CDMs [Pha+16]. Therefore, the service or content providers and operators are forced to use the DRM system of the Web browser they have implemented. The summary of CDMs supported by Web browsers is presented in Table 2.1.

Tab. 2.1.: Proprietary Content Decryption Modules (CDMs) of Web Browsers

Web Browser	Platform	DRM Executed in the CDM
Google Chrome	Android	Google Widevine
Apple Safari	iOS	Apple Fairplay [App17b]
Microsoft Edge	Windows	Microsoft Playready
Mozilla Firefox	All supported by Firefox platforms	Adobe Access [Ado17a]

The diversity of real-world implementations of CDMs, and therefore DRMs, and how contents are treated can be described through the following approaches: (1) Sandbox DRM and (2) Software Platform-dependent DRM.

(1) Sandbox DRM: in this scenario, the DRM scheme is implemented in a Pepper Plugin API (PPAPI) sandbox that restricts the data flows in and out of the CDM. In contrast to the Software Platform-dependent DRM approach, the sandbox runs in the same process as the browser. PPAPI is the Chromium's alternative to the deprecated Netscape Plugin API (NPAPI),

analyzed in Annex B.3.2.3. Such an implementation was presented by Mozilla Firefox in 2014 for Adobe Access DRM [A. 17].

(2) Software Platform-dependent DRM: this approach is a platform-dependent and proprietary DRM architecture. CDM is not implemented as a part of the Web browser. It can be realized either in software or in hardware-assisted software. In this case, the content is bundled not only to a particular EME-compliant DRM scheme but also to a platform.

Considering all these constraints, the operators and content providers support the non-proprietary development of these technologies. They also support a consolidation of the fragmented environments in this domain with such standards like MPEG-DASH with Common Encryption (CENC) [ISO16] and the previously-mentioned EME but with an option for exchanging DRM schemes. CENC is analyzed in detail in Annex B.3.2.2.

Simmons et al. in [J. 14b] [J. 14a] divide the issues that occur in Web DRM deployments into the following three groups:

- **Enhanced Security** - the requirements for enhanced content protection [Mot15] will force a shift in the execution of Web DRM out of the browser context and into the platform;
- **Interoperability of state** - as CDM is a part of the browser and is provided by a browser vendor, there will be issues with the inconsistency of the Web DRM state when transitioned to a different CDM. This will impact the Web DRM interoperability across platforms;
- **Licensing** - it is a crucial task to establish non-licensed CDM technology, as the majority of embedded devices depend on open source browsers.

Simmons et al. introduce the *Content Decryption Module interface (CDMi)* architecture to address all three issues. *CDMi* extends the EME specification towards the OS-level DRM. It shifts the DRM execution out of the browser context, integrating the DRM with the platform.

The CDM, in this approach, proxies the EME calls via the *CDMi* with Remote Procedure Calls (RPC) from a Web application to the DRM of the platform. The CDM does a logical mapping of methods or events specified in EME towards the *CDMi*. The (1) critical DRM functions are not executed in the browser context, (2) the standard *CDMi* interface enables state interoperability and (3) open source browsers must not need to incorporate any of the licensed components of the DRM of the platform.

In summary, in the variety of current EME deployments, e.g. for desktop browsers, the DRM runs in a CDM in the browser context. Implementations in the browser context were the first implementations for all platforms and still exist on the market.

In a joint research project, David Livshits[10] together with the author of this thesis, demonstrated an attack on Google's Desktop EME implementation for Widevine in June 2016. The high-level attack description and the video in which the attack is shown were published in a *Wired* article [K. 16] since, due to the Digital Millennium Copyright Act (DMCA) [Dig17], the rest of the information could not be revealed.

[10] David Livshits is a security researcher in the Cyber Security Research Center at Ben-Gurion University in Israel.

A proof-of-concept attack was demonstrated by utilizing a bug in Google's implementation that allowed access to the decrypted content stream. This shows that the EME protection of content requires advancements in its security. Also, a variety of browser security vulnerabilities has been demonstrated in the past, as analyzed in Chapter 2.1.3. It must be ensured that if the browser is compromised, it will not compromise the entire DRM system.

2.2.2.3. Ethical Summary

The developments of the Web towards the medium for media content delivery have enabled two significant changes in conventional content distribution schemes. On the one hand, new distribution techniques of media content have been enabled for copyright holders. These techniques, e.g. applications or services, arise from media formats that have been newly established on the Web, frameworks, and technologies developed for distribution and consumption of media content. On the other hand, a user can also become the owner of the content, as users have been enabled to store, copy and distribute the digital content on their own.

Interestingly, the latter development that enabled users to own the content, in conjunction with the fact that copying is a "*natural law of the digital world*" [B. 01], has killed conventional business models of content distribution. Therefore, the distribution methods, which at first seemed attractive and promising, also brought about the death of the content industry. Various media data, e.g. music, movies, books, etc., have become available for free on the Internet. The copyright holders were no longer capable of protecting their intellectual property.

However, Lefevre states that "*no organism capable of volitional action can escape the demands arising within it for some kind of property relationship. [...] The owner of property may rightfully take whatever steps he deems feasible and economically warranted to protect his property.*" [LeF71]

Here, *the first paradox* arises. A paradox lies in the natural laws of the digital world and human nature: the inability to make digital files uncopyable and intellectual property owners or copyright holders' natural need to protect their property.

To fulfill the need for protection, content protection systems have evolved. DRM was introduced to give media distributors control over their content even when it was in users' hands, in addition to a way of monetizing their content. It was also intended that DRM would equip customers with extended usage rights in comparison to CA systems[11].

Despite the initial aim to give the users the freedom to control purchased content, DRM has evolved into a technology that is constantly taking more freedom away. Here, *the second paradox* appears: a paradox lies between content sales and purchase contracts in the digital world. The media distributors want to sell their content. The more they sell, the more profit they gain.

However, even remembering that the whole Web is built around copying content, media distributors cannot allow users to copy and distribute purchased content, as the profit they

[11] CA is described in detail in Annex B.3.1, DRM is examined in Annex B.3.2. Here, only the general concepts are considered.

could make would be minimized. Therefore, they cut down the rights of users that purchased the content. This results in unfair purchase contracts that aren't visible to the users, but very beneficial for media distributors.

According to the conditions of the contract, which would become apparent to the user post-purchase, the user does not own the content, as multiple actions that fall under the definition of ownership are restricted. These include sharing content with others, storing the content on multiple devices, altering the content, etc.

Finally, users are frustrated when such purchase contracts are enacted. These and many other reasons could turn users against DRM [Fou] [ANS16] and to bypassing DRM protection [M4V]. To protect DRM technologies, the DMCA [Con98] prohibits the circumvention of content protection mechanisms. However, such a method for protection might not be the best path to choose. As Lefevre states, "*if protection is inadequate for any reason, there is still no justification for imposing some penalty upon the members of society at large.*" [LeF71]

The DRM controversy described above demands a rethinking and reconsidering of currently established DRM techniques. The core requirements of the new DRM models must be based on providing more flexibility and freedom to users. At first, it might seem to be a counterproductive step for media distributors with regard to sales profits. However, in the long term, this might be the only way to stabilize the media content market. Especially since more flexible and open DRM schemes already exist [CC09] [LW14] [Tha+15a].

The users could also utilize these schemes, as they also enter an era of digital media ownership. The use cases that arise here range from simple ones like picture sharing, up to very sensitive cases like smart homes and medical data sharing[12].

The Content Distribution Chain and its Future

Multiple content distribution models currently exist on the Web [Voo]. They are based on different underlying technologies and involve a variety of market players, such as content producers and creators, publishers and agencies, operators, customers, etc. The models that currently exist on the market could be generalized as presented in Figure 2.4, regardless of distribution technologies.

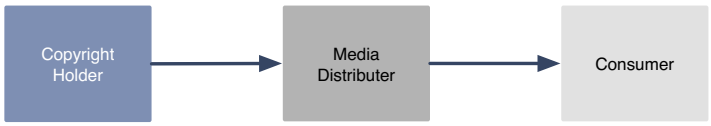


Fig. 2.4.: Current Model of Web Content Distribution

The Copyright Holders create the content. The Media Distributors distribute it towards the Consumers. This model emerged out of expensive and complicated delivery and distribution techniques that the Media Distributors deploy and thereby make their profit. One of the current platforms that could be an example of such a scheme available on the Web is Spotify [Spo17]. Customers pay their monthly subscription fees to Spotify's service, which plays a role of the Media Distributor.

[12] According to Dirk Thatmann, the research scientist at Service-centric Networking group at Telekom Innovation Laboratories, Berlin.

Spotify, in turn, works with various record labels that partially play the role of Copyright Holder. More precisely, they hold a sound recording copyright. The music creators are also Copyright Holders; they hold the music copyright itself. Although it might seem that the music royalties infinitely enrich the original creators of the music, in a real life scenario, it seems more like they don't reach the creators of the music at all with this distribution scheme [Pla]. In such a distribution scheme, the music royalties are usually split up between the Media Distributors and the Copyright Holders of sound recordings, e.g. record labels.

However, the latest developments on the Web have aimed to integrate media distribution into Web standards, enabling less expensive and less complex media distribution infrastructures.

These developments include efforts in establishing browser interfaces towards device middleware with media playback [W3C], integration in browsers of the media streaming interface MSE, examined in Chapter 2.2.1.2, and the interface for playback of encrypted content EME analyzed in detail in Chapter 2.2.2.1.

The industry initiative WAVE [Con16b] is aiming to establish interoperability between media streaming technologies and to enable a media playback runtime environment. Through such significant developments, the Media Distributors will lose their significance in the future.

The content distribution model will also be simplified towards the one presented in Figure 2.5. The Copyright Holders will be able to establish distribution on their own, therefore bypassing the Media Distributors. They will be responsible for the creation of content and its distribution, as the need for complex distribution technologies disappears.

Fig. 2.5.: Model of Web Content Distribution in the Future

Tier3 Media Distribution

The constant developments on the Web and corresponding streaming technologies have a significant impact on streaming infrastructure. The time of Tier1 media distributors is running out. Tier1 distributors are big operator companies, e.g. Cablecos, Telcos, etc., that were aggregating content from different content studios and distributing it to the user.

Now is the time of Tier2 distributors. These are companies that are delivering their content OTT. The OTT providers, such as AmazonTV, Netflix, Spotify, etc., are using the Web to deliver media content to end-users.

Tier3 media distributors will arise with the simplification of Web streaming technologies, which will be available as open-source on the Web[13]. Tier3 media distributors will be the Copyright Holders themselves, e.g. music creators, soccer clubs, etc. This model is a decentralized model of distribution, and it already exists on a small scale on social media.

[13] According to John Simmons, Media Platform Architect in Microsoft.

This also recalls the *Superdistribution* model predicted by Ryoichi Mori [Mor89] [MK90] [MK97]. One of the technologies that could fulfill this model might be the blockchain technology for distributed content protection [Fuj+15], [Kis+15].

Web DRM

The introduction of W3C EME API has provoked quite a bit of controversy, as Web standards are open and non-commercial. However, high-value content cannot be distributed without DRM, which currently makes DRM a necessary market requirement.

It is also important to note that EME is not DRM. EME is an interface enabled in browsers to natively decrypt encrypted content, which aims to simplify the distribution of high-value content on the Web in a standard way.

2.2.3 Trusted Execution Environments

As stated in Chapter 2.2.2.2, the latest developments in Web DRM require measures to improve protection through advanced security techniques. These requirements are also aligned with the recent developments in the media industry: advanced security for content protection is also required by the guidelines of MovieLabs [Mot17], the research lab driven by the six major motion picture studios.

As stated in the guidelines [Mot15], for a robust content protection of high-value content "*the platform shall support a secure processing environment isolated by hardware mechanisms running only authenticated code for performing critical operations.*" Thereby, with the advent of high-value content, like Ultra-HD and beyond, the advanced security of EME might become the only solution approved by studios. As enablers of advanced levels of security, tamper-resistant architectures for code isolation, i.e. TEEs, are analyzed in this chapter.

GlobalPlatform [Glo17a] has established TEE specifications, that are "*regarded as the international industry standard*" [Glo17b]. These specifications of TEE architecture and multiple APIs are widely accepted in the industry. GlobalPlatform identifies TEE as an "*execution environment that runs alongside but isolated from a device main operating system*" that satisfies the following requirements: "*protects TEE assets from general software attacks, defines rigid safeguards as to data and functions that a program can access, and resists a set of defined threats*" [Glo17c].

Being rather general and imprecise, this definition, like many other current definitions [Gar+03] [Ope09] [Vas+14] is very inconsistent. Therefore, in 2016 works appeared in academia about a re-definition of TEE. Solutions for the re-structuring and re-systemization of TEE have been present in academia for the last decade [Sab+15a] [Nga+16] [Sab+15b].

This thesis follows the systematization presented by Sabt et al. [Sab+15a] that categorizes TEE as a so-called Dual Execution Environment (DEE) approach. TEE builds upon the following core functions [Sab+15b] [Ekb13]:

1. **Secure Boot -** it is important to ensure the integrity of TEE through a secure boot. The bootstrapping of the system is interrupted when modifications in measurements[14] throughout bootstrapping are detected.

[14]The measurements are the cryptographic hashes that are computed for the following bootstrap code.

2. **Isolation -** TEE separates the platform into compartments: a secure environment referred to as a secure OS, where sensitive data are processed, and non-secure environment referred to as a non-secure OS, where a rich OS is executed. The secure OS makes tamper-resistant functions possible. If the non-secure OS is compromised, the secure OS must not be impacted. To create this, a strong isolation between those two compartments is crucial. Also, storage in both compartments must be isolated from one another, enabling access to secure storage only to authorized parties.

3. **Inter-Compartment Communication -** the communication between secure and non-secure compartments must be strongly monitored and follow a securely defined interface.

Various tamper-resistant software and hardware architectures have been presented for the protection of systems in the past. Some of the approaches, e.g. AEGIS [Suh+03] [Suh+05], are hardware architectures that would require significant changes in processor architectures. The secure microkernel approaches, e.g. SeL4 [Kle+09], enable fundamental functions of an OS with only a minimum amount of code reducing the Trusted Computer Base (TCB). The only software that is executed in the privileged mode is a secure microkernel. The majority of these architectures require modifications in the OS and therefore in existing applications.

To address these complications, the DEE approach [Sab+15a] has been introduced. This approach enables the execution of secure and non-secure OSs on one platform. The non-secure OS is the fully equipped, rich OS, and the secure OS is the OS with limited functionalities that runs sensitive data. Compared with tamper-resistant architectures presented before, the DEE is considered to be a very practical approach that has found multiple fields of application [Hus+05] [Wil+07] [Kos+09] [Fre+10] [Gud+11] [Pin+14] [San+14].

For empowering isolation, the fundamental concept of the DEE is the Separation Kernel (SK) [Sab+15a] [Rus81] [AJ+83] and it "*assures the property of isolated execution*" [Sab+15b]. The SK enables the coexistence of OSs with different security levels on one platform. In other words, SK establishes partitions isolated from each other, with a strictly controlled interface for communicating between these partitions.

This thesis follows the definition of the TEE as a "*tamper-resistant processing environment that runs on a separation kernel. It guarantees the authenticity of the executed code, the integrity of the runtime states (e.g. CPU registers, memory and sensitive I/O), and the confidentiality of its code, data and runtime states stored in a persistent memory. TEE shall be able to provide the remote attestation that proves its trustworthiness for third-parties. The content of TEE is not static; it can be securely updated*" [Sab+15b].

2.2.3.1. Isolation in Dual Execution Environment

According to Sabt et al., the ways this isolation might be implemented in the context of the DEE are systematized as follows:

1. Isolation through dedicated hardware - the platform is extended with an integrated circuit of a smart card or equipped with a dedicated secure co-processor that is physically isolated from the non-secure OS, and therefore has separated memory. The dedicated hardware runs the secure OS.

2. Isolation through bare-metal hypervisors[15] - the platform executes a hypervisor in the most privileged processor mode. The hypervisor can then host multiple OSs on the same platform. Here, the hypervisor takes on a role of the SK.

3. Isolation through processor extensions - the processor is enhanced with hardware extensions that can execute sensitive code securely within a non-secure OS.

The first approach is a dedicated cryptographic processor that provides secure non-volatile memory and a bootstrap loader that enables key generation and key handling. One of these technologies is the secure co-processor architecture Trusted Platform Module (TPM) specified in [Tru17b] by the Trusted Computing Group (TCG) [Tru17a].

TPM, in addition to the general features of such approaches provides functions such as an authenticated boot[16]. TPM, unlike the smart cards, cannot be removed and its TCB is bound to the hardware platform [Ekb13]. If the dedicated hardware is not present, the TPM may also be virtualized and implemented in software [Zha+07].

However, the software TPMs have yet to find extensive real-world applications. As stated in [Wil+07] the additional silicon costs for a co-processor might limit its resources and degrade its performance.

Such technologies like Intel Trusted Execution Technology (TXT) [J. 17] build upon the TPM and provide a software-based isolation of sensitive code. They also address some of the TPM drawbacks, like the fact that it's impossible to repeat measurements after the system has booted in the authenticated boot.

In 2013 Intel released a new extensions architecture called Software Guard Extensions (SGX). This design allows for an isolation of code in the user space in a hardware-assisted manner[17]. Isolated containers in SGX are called enclaves. Hoekstra et al. present some of the use cases of SGX utilization in [Hoe+13], e.g. one-time password or secure video conferencing.

In the second approach based on bare-metal hypervisors, a hypervisor implements the same Instruction Set Architecture (ISA) as platform hardware. Multiple OSs that might run on such hypervisors are entirely isolated from each other.

In ARM-based[18] smart devices support for ARM full-virtualization[19] is still limited as this technology has been introduced only in ARMv7 [DN14].

The third approach is presented by ARM TrustZone (TZ) processor extensions. TZ implements a unique processor virtualization approach, where two virtual processors execute secure and non-secure OSs in an isolated manner [ARM09a]. TZ is analyzed in-depth in the following chapter.

[15] Bare-metal hypervisors access the hardware resources directly and do not require any OS installed on the hardware platform.

[16] The authenticated boot throughout the booting process records measurements' values throughout a booting process. It utilizes Platform Configurations Registers (PCRs) and performs Root of Trust Measurement (RTM). Thus, any arbitrary state, in which the platform boots, is recorded and can be reported [PB03].

[17] For example, the platform hardware controls the fact that enclaves start at authorized locations. Unscheduled enclave exits are controlled on the subject of data leakage [McK+13].

[18] Smart and embedded devices based on ARM processors have the biggest share of the market [ARM13].

[19] It requires full virtualization of the hardware platform, and therefore the guest OS does not have to modify its kernel.

2.2.3.2. ARM TrustZone

ARM TZ is available on ARM processors since the release of ARM-1176J-S [ARM09b]. ARM addresses the DEE concept by introducing two virtual CPUs (vCPUs) that execute non-secure and secure OSs in isolation from each other. TZ [ARM17a] [Sti16] extends the standard execution of the non-secure OS that is complex and therefore potentially vulnerable with an additional *Secure World* mode. The *Secure World* mode executes the secure OS and has a separate memory for sensitive data and code.

The execution of the non-secure OS is done in the *Normal World*. The processor with ARM TZ always boots in the *Secure World*, allowing for a secure boot of the system. Applications executed in the secure OS are called *Trusted Applications (TAs)*. The two modes are connected by a special processing context - the *Monitor Mode*. The *Monitor Mode* executes the *Secure Monitor*. The *Secure Monitor* implements the SK in the context of DEE and controls the switching between the OSs.

The switch is executed via a system call or a *Secure Monitor Call (SMC)* requested at the *Secure Monitor* that strictly controls access to the *Secure World*. The *Secure Monitor* triggers a software interrupt after verifying whether the *Secure World* entry can be granted or not, based on device and processor settings. In the case of successful verification, the *Monitor Mode* clears the *Non-Secure (NS)* bit that operates the entrance into the *Secure World*.

The function of changing the *NS* bit can **only** be executed by the *Monitor Mode*. The *NS* bit signals to *Secure World*-aware hardware that the platform is running in *Secure World*. The hardware logic, e.g. clocks, timers, controllers, etc., depend on the *NS* bit and change operation modes correspondingly.

ARM TZ-based TEE is widely deployed in practice and also used extensively in academia [San+14] [Hus+05]. TZ applications in academia will be presented in Chapter 2.4.2.2. In practice, the implementations based on ARM TZ are presented through the following technologies: proprietary, open source or based on an industry standard. Established companies implement proprietary-developed TEEs in their devices, e.g. Samsung implements TrustZone-based Real-time Kernel Protection (TZ-RKP) [Sam16a] for Samsung Galaxy, Nokia - now Microsoft - uses On-board Credentials (ObC) [Kos+09] for Nokia Lumia.

Closed source TEEs also include Trustonic TEEs [Tru17c] or Qualcomm's Secure Execution Environment (QSEE) [Qua17]. Sierraware [Sie17a] offers its TEE implementation, called SierraTEE, [Sie17b] as open source. The first and only organization that has made efforts to establish an industry standard for TEE is Linaro. Together with STMicroelectronics [STM17] and other partners, they position their open source implementation Open Portable TEE (OP-TEE) [Lin17c] as a reference implementation for TEE, and offer it as open source.

2.3 Interactive Services

Interactive media services coming from the broadcaster domain are the focus of this chapter. Chapter 2.3.1 analyzes how interactive services emerged and which standards play a role in the formation of interactive media.

The most accepted and widely deployed interactive TV standard in Europe, the Hybrid Broadcast Broadband TV (HbbTV) is analyzed in Chapter 2.3.2.

Chapter 2.3.3 focuses on the technology of signaling in HbbTV applications. Chapter 2.3.4 analyzes the currently known vulnerabilities in the HbbTV standard as presented in academia.

2.3.1 Interactive TV

When they reached the TV domain, Web technologies began to change the TV landscape. This created new opportunities for interactive services. A TV system is called an interactive TV system, if audio, video, and text data is delivered over traditional TV broadcasts, while a broadband Internet connection is used as a channel to support interactive services. These systems are referred to as Hybrid TV systems, as they feature hybrid capabilities due to the fusion of broadcast and broadband delivery. The broadband Internet connection is an interactive channel or a back channel.

MHEG-5 is the first standard for interactive TV, developed by the ISO Multimedia and Hypermedia information coding Expert Group (MHEG) [MHE16]. It is specified in the ETSI standard 202 184 [ETS10e]. The MHEG-5 standard enables the broadcast and presentation of interactive applications by using its object-oriented programming language MHEG-5. MHP implements Java Applets to achieve user interactivity.

Furthermore, the DVB project developed middleware specifications to provide interactivity for DVB services. Herewith, a Java-based API was specified to run applications downloaded from DVB broadcast receivers from different manufacturers. This Java-based API is known as Multimedia Home Platform (MHP) [ETS10b]. None of these technologies have been widely deployed. MHEG-5 had very limited interactive capabilities and MHP, being rather a complex system, has never gained popularity due to several legal issues.

2.3.2 Hybrid Broadcast Broadband TV Standard

At the same time, a new initiative to standardize Hybrid TV was begun by several broadcasters in Europe [Hyb16]. This led to an open standard in 2009: Hybrid Broadcast Broadband TV (HbbTV).

HbbTV combines Web-based technologies with a DVB-based video delivery analyzed further in Chapter 2.3.3.1. ETSI approved version 1.1.1 of the HbbTV specification in 2010 and published it as ETSI TS 102 796 [ETS10d]. This specification is commonly known as HbbTV 1.0.

The HbbTV specification version 2.0 was updated in October 2015 [ETS15]. HbbTV is based on the existing standards presented in Appendix B.4.1. Appendix B.4.3 also analyzes the inconsistency of HbbTV devices and the device fragmentation that's thereby emerging in the HbbTV market.

Analog to the DVB Standard for Digital TV (DTV) in Europe, similar standards were specified in other countries: ATSC [Adv16] mostly in North America, ISDB-T [Int16] in Japan and South America, DTMB [J. 15b] primarily in China.

2.3.3 Processing of HbbTV Applications

Over the Broadcast Interface, AIT data, DVB-compliant media content, application data and stream events are received by an HbbTV-enabled client, i.e. a Hybrid Terminal, as depicted in Figure 2.6. Media data in the context of HbbTV are both linear data, i.e. Live TV, and non-linear data, e.g. VOD services.

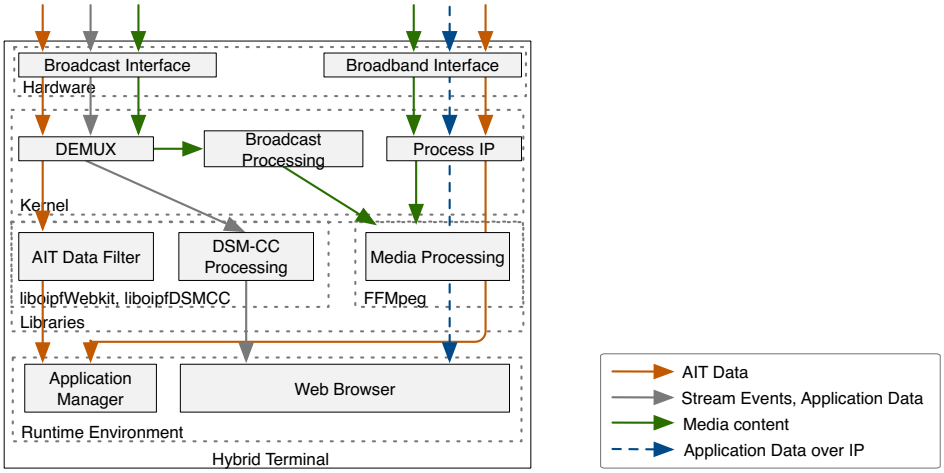


Fig. 2.6.: High-level Architecture of the HbbTV Terminal (adapted from [ETS15])

The DSM-CC Processing recovers data from the DSM-CC object carousel analyzed further in Chapter 2.3.3.1, and makes it available for the Runtime Environment. The AIT Data Filter filters the AIT data and makes them available for the Runtime Environment. In the Runtime Environment, which consists of Web browser and an Application Manager, the interactive applications are composited and executed. The customized, i.e. HbbTV-enabled, Web browser is used as the run-time environment for the HbbTV applications.

The Application Manager manages the application's life-cycle. The Broadcast Processing component processes the broadcast media content. This content is subsequently processed by the Media Processing component that scales and embeds the content into applications. Over the Broadband Interface, non-linear content and the application data[20] can be requested.

The signaling and delivery of HbbTV applications are specified in TS 102 809 [ETS13]. The HbbTV standard defines two different types of HbbTV applications: broadcast-related and broadcast-independent applications. The broadcast-related applications are presented further in Chapter 2.3.3.2, broadcast-independent applications are analyzed in Appendix B.4.2.

2.3.3.1. Digital Video Broadcasting

The DVB Project was founded in 1993 to develop an open system for presenting MPEG-2-based DTV. The DVB Group [Dig16b] brought broadcast operators, radio standardization committees, and manufacturers of consumer electronic equipment together. MPEG-2 is a Moving Picture Experts Group [Mov16] standard for digital A/V transmission and compression algorithms.

[20]The application data can be received by both broadband and broadcast interfaces.

Digital Storage Media Command and Control Because of the one-way nature of data transmission in broadcast channels, the broadcast receiver cannot request any content from the transmitting station. To imitate interactivity for broadcast channels, the Digital Storage Media Command and Control (DSM-CC) protocol was specified. The DSM-CC protocol is the application protocol that provides essential control functions and specific operations to manage MPEG-2 bitstreams on digital storage media.

EN 301 192 [ETS08a] specifies data and object broadcast technology over the DSM-CC Data and Object Carousel, respectively. The data carousel enables broadcasters to transmit so-called modules of data to the receiver. The object carousel is an extension of the data carousel and provides broadcasters with a file system-like functionality. The described DSM-CC modules are broadcasted in a loop, one after the other, and the receiver has to wait for the module containing the required data.

MPEG-2 Transport Stream The MPEG-2 Transport Stream (TS) [ISO94] consists of transport packets, each with a length of 188 bytes: 4 bytes for the header and 184 bytes for the payload. This transport stream delivers the following types of information: audio, video, data, or Program Specific Information (PSI). The MPEG-2 TS thus has multiple interleaved elementary streams, each of which may transport either audio, video, data, or metadata, i.e. PSI. The packets are identified by a Packet ID (PID), which is transmitted in the header. All packets with the same PID belong to the same elementary stream.

The PSI includes the following tables, as defined in the MPEG standards: Program Association Table (PAT) and Program Map Table (PMT). The PAT in a TS is always identified by `PID 0`. An exemplary PAT and PMT structure is illustrated in Figure 2.7.

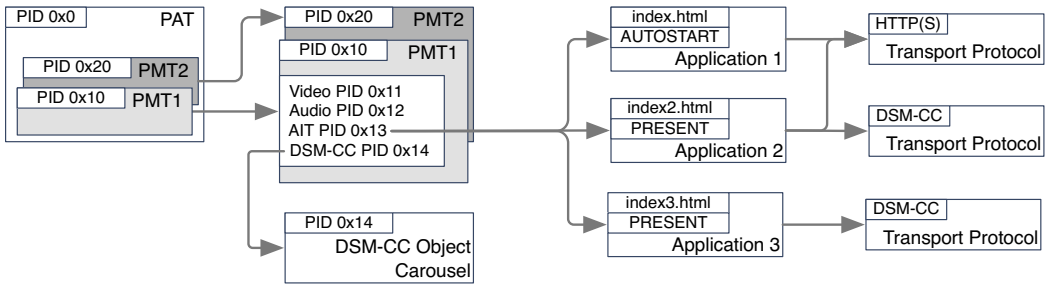


Fig. 2.7.: Structure of Broadcast-related Application Signaling: Program Association Table (PAT) and Program Map Table (PMT) Structure

The PAT lists MPEG-2 TS programs and corresponding PMTs: `PID=0x10` for PMT1. The PMT lists the PIDs of elementary streams belonging to a program: `Video PID=0x11`, `Audio PID=0x12`, Application Information Table (AIT) [ETS13] [ETS15] `AIT PID=0x13`, and `DSM-CC PID=0x14` for PMT1. AIT is a service table that includes references to applications, i.e. application addresses, through which the applications are downloaded and executed on the end-device.

A PMT can list many AITs and therefore multiple applications. In the example presented in Figure 2.7, the AIT contains references to three applications: `index.html` as a reference to Application 1, `index2.html` for Application 2 and `index3.html` for Application 3.

Application 1 is identified as an AUTOSTART[21] application, which means that it will be automatically started once the channel is selected, or after an AIT update. The two other applications are PRESENT applications, which means that they will *only* be started once activated by an end-user.

An application must provide at least one transport protocol descriptor, HTTP(S) or DSM-CC object carousel. In this example Application 1 provides the protocol descriptor HTTP(S), Application 2 provides two protocol descriptors HTTP(S) and the DSM-CC object carousel, Application 3 provides the DSM-CC object carousel as a descriptor. If the former is indicated, the application will be fetched from a Web server using HTTP(S). Otherwise, the application is provided within the DVB MPEG-2 TS, in the DSM-CC object carousel.

2.3.3.2. Broadcast-Related Applications

Applications associated with broadcast services, or events within a service, are broadcast-related applications. These applications are called HbbTV Applications. They are signaled over DSM-CC with the AIT, using the broadcast connection. Application signaling in HbbTV is referred to as Red-Button signaling, as the end-user activates the applications by pressing a red button on a remote control.

DVB Signaling and DVB/HTTP Delivery The signaling of AIT broadcast data over DSM-CC is depicted in Figure 2.8. The AIT, which is included in the MPEG-2 TS, is extracted by the demultiplexer.

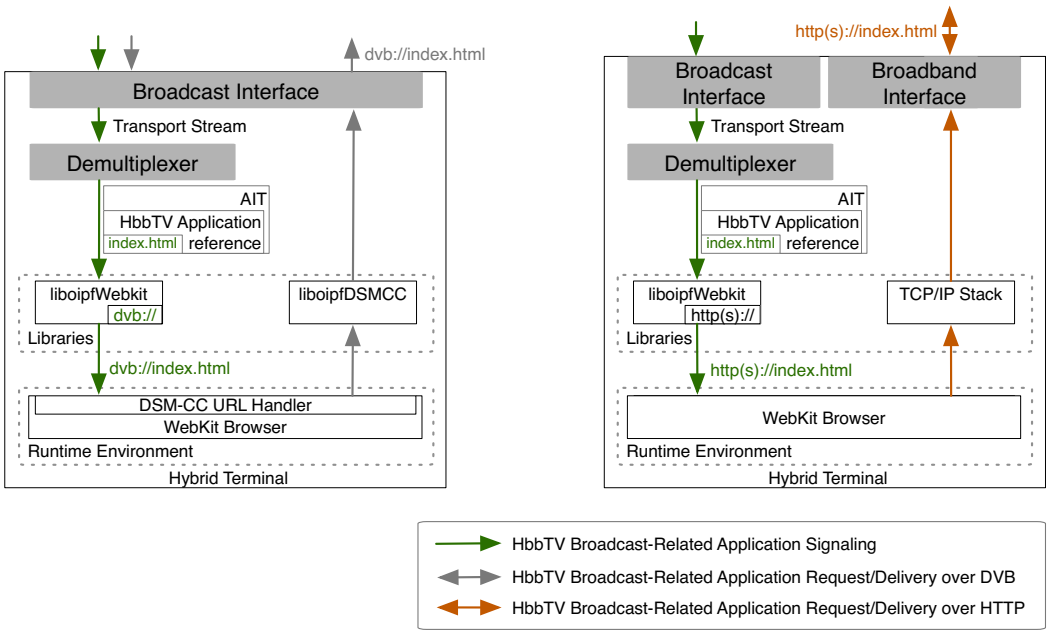


Fig. 2.8.: HbbTV Application Signaling over DSM-CC in DVB: HbbTV Application Delivery over DSM-CC in DVB (left) and over HTTP(S) (right)

[21] After being started, AUTOSTART applications present no UI and display the *Red Button* notification to inform the end-user of application availability. By pressing the red button on a remote control, the end-user would activate the application.

First, the data object is extracted. The AIT contains a reference to the HbbTV Application's object `index.html`. This object is processed by the `oipfWebkit` library, which adds the `dvb://` prefix to the reference for DVB delivery, as demonstrated on the left side in Figure 2.8. The data itself is received by the HbbTV terminal: the `dvb://index.html` URL is accessed by the Webkit browser, which uses the DSM-CC URL handler to fetch the object. The `oipfDSMCC` library implements the DSM-CC URL Handler.

To download the `index.html` object, the URL handler has to wait for the `index.html` object to become available from the DSM-CC object carousel in the MPEG-2 TS. The `dvb://` prefix is added because the application data itself would be received through the DSM-CC carousel, over the broadcast interface. Broadcast-related applications can be requested and delivered over both broadcast and broadband connections.

With the `http(s)://` prefix added, application data is requested by the TCP/IP Stack over the broadband interface, as shown on the right side in Figure 2.8. When broadband delivery is executed, the AIT contains an HbbTV Application URL for an HTML Web page provided by the broadcaster. The HTML code is then downloaded and interpreted by the terminal via the broadband interface.

2.3.3.3. Alternative HbbTV Signaling

HbbTV has begun to work on the specification of Independent Application Discovery (IAD). They address the scenario in which application signaling does not reach the HbbTV terminal. This happens in the case where service operators do not activate Red-Button signaling, or the TV is connected by HDMI or SCART.

Three options for solving these issues are considered: (1) in-band transmission of the AIT, which then would be extracted by the terminal, (2) in-band transmission of the URL, which then points to the AIT and (3) broadcasters signaling a special server ID, which is then used by the terminal to resolve the server that serves the AIT. Currently, the third option is preferable and may become a part of the standard. However, it is still unclear whether broadcasters will accept it or not.

2.3.4 HbbTV Insecurity

As discussed in Problem 5.2 in Chapter 1.3, security research groups have proven both theoretically and practically the insecurity of the combination of DVB with Web technologies in HbbTV [Ghi+13] [GT14] [OK15] [Mic15]. The large-scale attack on Smart TVs, presented in [Mic15], was made possible through the following system vulnerabilities:

- The broadcasters in DVB do not have any means to authenticate themselves for DVB receivers. Therefore, the data generated by broadcasters and sent to the receivers could easily be forged by attackers. The most critical data here, as previous research has shown, are the HbbTV Application URLs. By forging those, attackers might direct traffic to any malicious resource on the Web.
- HbbTV Applications are pre-loaded and executed in the background without any active trigger performed by users. This might enable remote triggering by malicious parties and uncontrollable execution of the forged applications.

The security issues exposed by those attacks have raised awareness within HbbTV and DVB groups. This resulted in the definition of security countermeasures to mitigate those kinds of HbbTV-assisted attacks in the future. Both HbbTV [Hbb15] and DVB [J. 15a] have announced an active process of ongoing specification work.

The preferred countermeasures are based on suggestions made towards HbbTV in [Mic15] for the use of Certificate Authority (CA) based on X.509 v3 certificates. As stated in [Hbb15], the approach is to add extra tables to the elementary streams that would contain (1) signatures for the existing AIT and DSM-CC sections in these streams and (2) certificates containing keys for the validation of those signatures.

The certificates are to be either regular PKI certificates issued by the country of the broadcaster, or persistent broadcaster self-signed certificates, where trust is to be based on the overall time in which certificates have been seen in the broadcast.

2.4 Related Work

This section analyzes the publications in academia that are most relevant for the thesis with regard to SRSE environments examined in Chapter 2.4.1, its services, and its security. The services considered in this analysis are the protected streaming and interactive services that are examined in Chapters 2.4.2 and 2.4.3 respectively.

2.4.1 Remote Service Execution and its Security Threats

Chapter 2.4.1.1 presents RSE systems and architectures that are currently being researched in academic contexts. Afterward, given that the computation power in RSE resides in the Cloud, an overview of well-established and also potentially new security challenges for virtualized environments is presented in Chapter 2.4.1.2. These security challenges apply to all Cloud environments, and therefore pose security threats to RSE, which could jeopardize a WMS platform.

2.4.1.1. Remote Service Execution

In their work, Zhu et al. [Zhu+11] address the problem of scalability in server-based pre-rendering systems. These are RSE environments that overtake rendering functions from clients. The work proposes a peer-assisted rendering concept to address the problem of scalability. This problem arises due to high server loads while executing computationally expensive operations like rendering.

The authors also address a latency problem that occurs when shifting rendering to the Cloud. Authors display the feasibility of peer-assisted rendering for online gaming within a virtual scene that is shared between multiple mobile terminals.

Lee et al. present a virtualization system for Web applications to run these applications on legacy STBs [Lee+12]. The authors shift a Web-based IPTV middleware or so-called Web middleware to a remote Cloud server. Web applications are executed on the remote server and then streamed down to legacy STBs as a video stream.

Chun et al. present the CloneCloud framework that virtualizes mobile applications through their execution in the Cloud [Chu+11]. The system consists of an application partitioner that allows for the distributed execution of unmodified mobile applications on mobile devices and in the Cloud.

The tasks that could be executed faster on a remote host are migrated to the Cloud in a flexible manner and performed in an application-level virtual machine. They are then migrated back to the mobile client and merged with or re-integrated into the clients' tasks. Therefore, mobiles could seamlessly offload parts of execution tasks to the Cloud to speed up execution and increase battery lifetime. This work is based on the authors' vision that had been published earlier in [CM09].

The Media Cloud approach [DS+11] proposed by D. Diaz-Sanchez et al. deals with media distribution beyond the boundaries of a local network. The local network in this work interoperates with DLNA [Dig16a] and UPnP [Uni16]. The extension of the local network through a home gateway application serves for sharing content in a home domain and on the Cloud. In this approach, rendering, service and composition tasks reside on the client side.

Liao et al. [Lia+16] address the problem of high server load in Cloud gaming systems. Cloud gaming systems are RSE architectures, where servers are responsible for game execution and graphic content rendering. Game contents are delivered to clients by video streaming, therefore posing a high server load. For this reason, an alternative approach to graphics streaming was developed in [Jur+09] that shifts some of the computations back to the client for local rendering. However, as the authors state, this approach consumes a large amount of bandwidth. Therefore, they propose a new LiveRender system based on compressed graphics streaming. This system utilizes compression techniques to save data transmission without any impact on video quality.

Suksomboon et al. examine [Suk+15] virtualization options for Customer Premises Equipment (CPE) in detail. They question the worthiness of CPE-function virtualization and develop a determination framework that uses heuristic algorithms: a modified Karger's algorithm [KS96] and an algorithm proposed by the authors that takes virtualization costs into account. This framework can determine the functions that should be shifted into the Cloud and the functions that should reside on the CPE.

In their work, they first transform standard CPE functions into a functionality graph. Then, a virtualization determination process is applied to the defined graph. Regarding system security, Suksomboon et al. pose restrictions on the placement of content protection functions. These must be executed locally and therefore, for this work, they are beyond the scope of the virtualization determination process. Finally, the modified Karger's algorithm determines that none of the CPE functions are worth virtualizing. However, the algorithm proposed by the authors indicates some functions that should be virtualized depending on the use case.

2.4.1.2. Virtualization and Cloud Security

Concerning Cloud security in general, numerous security issues arise, as categorized in [Cho+09] by R. Chow et al.: traditional security, availability and third-party data control. Their work presents an overview of potential access management problems in Cloud envi-

ronments. Noteworthy problems include increased authentication demands and mash-up authorization - an authorization against the services, which perform data mash-ups.

The authors of the paper recognize some advantages in the advent of thin clients - clients in which the computation is mostly taking place on the server-side. This will make software piracy more difficult, and it will also centralize user monitoring for content providers. In this case, the authentication must be secured to avoid stealing of authentication credentials through phishing or other techniques.

Concerning increasing data mash-ups, it is evident that their authorization mechanisms must be improved to avoid data leaks. The authors propose an information-centric approach to security, where the data will be protected not from the outside, but from within. This will be achieved by putting intelligence into the data itself.

Schoo et al. demonstrate [Sch+11] the main threats to Cloud networking in their position paper. It's mostly virtualization environment threats that are highlighted in this paper. To prevent such threats, isolation between Virtual Machines (VM) must be ensured. To prevent data thefts, a malicious use of hypervisors must also be avoided, as the malicious machine could read the packets dedicated to another machine. Authors indicate that trust models must also be ensured, as there is a chance that malicious VMs will get control of the hypervisor and therefore the physical hardware of a server farm.

With regard to a misuse of hardware virtualization by untrusted VMs, multiple works have been published on Virtual Machine Based Rootkits (VMBR). VMBRs install a virtual machine monitor underneath an original OS, turning this system into a virtual machine. The VMBR malware is difficult to detect as the original OS cannot access their states. King and Chen present proof-of-concept attacks with SubVirt VMBR [KC06] and Rutkowska and Tereshkin with Bluepill [RT08].

Chen et al. examine Cloud computing security threats and state that only some of them are truly justified concerns [Che+10]. The authors highlight that many such security concerns are possible over-reactions. The authors consider such issues as phishing, data losses, weaknesses in passwords, compromised hosts and virtual machine vulnerabilities as well-established challenges. They come up in related security fields like Web security, data assurance, and VM security.

In contrast to these already known issues, authors identify a new field arising in Cloud computing security - unexpected side channels and covert channels attacks created in the shared resource environments of the Cloud. For example, Ristenpart et al. prove that virtualization is not always secure [Ris+09]. The authors implement a side channel between two VMs on the same physical hardware, in which one of these VMs is compromised.

This makes an SSH keystroke timing attack on SSH possible, as presented earlier by Song et al. [Son+01], where even very simple statistical techniques could reveal a password's length or even a root password. This is due to the design of the SSH protocol, in which every single keystroke typed by a user is sent to a remote machine in a separate IP packet right after the user has pressed the symbol.

2.4.2 Protected Streaming

The security functions for RSE that are part of the research in this thesis are the encryption and decryption of video content. These functions are referred to as content protection and are examined in detail in Chapter 4.1.1.

Chapter 2.4.2.1 analyses the virtualization approaches to content protection presented in academia and the vulnerabilities associated with them. Some architectures that secure client platforms are demonstrated in Chapter 2.4.2.2. Chapter 2.4.2.3 presents possible threats to such architectures by analyzing attacks on these architectures as presented by researchers.

2.4.2.1. Virtualization of Content Protection and its Security

Ghodke and Figueiredo have identified DRM system vulnerabilities in virtual environments [GF04]. In their implementation, the authors prove that, in cases where DRM clients are not aware of execution within a virtual machine, DRM usage rules cannot be enforced.

The experiments, conducted by the authors, prove that copies of DRM-protected audio data can be made within a guest OS, even though it is not permitted by a DRM system and the DRM is enforced in guest OS drivers. Therefore, the authors conclude that DRM-clients must be provided with techniques for execution detection - whether it is executed within a VM or on physical hardware.

Lee et al. enable content protection in the Cloud with a DRM-as-a-Service system [Lee+13]. The authors address the need of service providers to support multiple DRM systems with the help of a Cloud DRM system. The Cloud DRM system offers such functions as content packaging and license management, key and domain management. The authors discuss the Cloud DRM system by defining the architecture and service use cases for such a system.

Zou and Petrlic also partly address Cloud-based content protection mechanisms. In their work, Zou et al. present a mobile DRM system that utilizes a sim card on a mobile device for key storage [Zou+10]. This makes the sim card-based DRM system more flexible than device-based DRMs, as the latter depend on unique, globally defined, device identifiers and secure players. Moreover, a sim card-based system is cheaper than a smart card-based one due to the additional costs for smart card readers.

The authors utilize Cloud computing paradigms to enable Cloud-based DRM back-end for efficient management of unstructured DRM data for images, e-books, audio, and video. Petrlic addresses user privacy issues when applying DRM to restrict software usage in the Cloud computing era [Pet12]. The author improves the proxy re-encryption schemes presented earlier and creates ciphertexts that are indistinguishable in his proposed scheme. This provides anonymity for users and reduces the possibility of users' profile building by any party.

Villegas et al. address in [Vil+12] the appearance in CA, also in network-based CA, of the problem of key leakage from legal to illegal devices with permanent bidirectional connectivity. The authors propose a forensic mechanism that detects the source of the leakage, assuming that the leakage source could only be a legal device. The mechanism is based on over-encryption of the keys with group-assigned keys that could lead back to the leaking STB after the keys have been illegally published. To create complex client decision logic without

expensive hardware, the authors of the paper enable a so-called network smart card. This is a server in the Cloud that stores cryptographic information and provides keys to authorized clients.

2.4.2.2. Trusted Execution Environments for End-Device Security

Aciicmez et al. present a general framework for trusted DVB STBs that supports compartment isolations through secure software to enable multiple DVB CA systems on a single system [Aci+09]. The secure DVB STB approach presented by the authors is based on off-the-shelf trusted computing technologies. It provides both service providers and users with secure scenarios for accessing protected content. The authors provide a trusted infrastructure that supports virtualized TPMs with secure boot and remote attestation possibilities for CA system providers.

Thekkath at al. examine an architecture of an execute-only memory (XOM) hardware implementation [The+00]. XOM allows instructions saved in memory to be executed but not manipulated. The XOM machine presented by the authors is based on internal, separated compartments that isolate independent software applications. Processes from one compartment cannot access the data from processes in another compartment, as cryptographic techniques and special rules are applied. The authors indicate that it is possible to create a multi-tasking machine where almost all applications run in XOM mode. A virtualization of the XOM approach would be applicable but not realistic due to efficiency drawbacks.

Suh at al. demonstrate a hardware design for single-chip secure processor AEGIS. In contrast to non-volatile memory approaches, AEGIS utilizes Physical Unclonable Functions (PUFs) to generate and protect unique secrets that, as the authors state, makes the processor secure against both physical and software attacks. This work originated in 2003 [Suh+03] and was implemented later on with PUFs in 2005 [Suh+05]. The authors enable application execution attestation through proofs available for a system user and remote parties.

To certify the initial state correctness of an application, the proofs are combined out of both a hash computed during the execution program, and a unique processor secret. To validate the final result, the system performs a verification of values coming from off-chip memory and encrypts on-chip memory. Thus, the entire memory can be considered secure. Within the AEGIS architecture, the application must not be executed in Processor Secure Environments (PSE) at all times. The PSE execution can be suspended, and then the unprotected execution does not have access to any data in PSE.

Costan et al. extend a standard RISC architecture to enable isolated software modules [Cos+15]. The modules share hardware resources and run concurrently. Sanctum APIs are comparable with SGX APIs to some extent. However, Sanctum addresses some of the attacks as memory cache and timing access pattern attacks, to which SGX remain vulnerable.

ARM TrustZone Technology

Santos et al. present a concept for a Trusted Language Runtime (TLR) for the protection of .NET mobile applications [San+14]. The TLR system uses ARM TrustZone technology to isolate an application from the OS and other applications. TLR enables the separation of application logic into security-sensitive and non-security-sensitive by introducing the trustbox and trustlet primitives. This allows small application components to be executed in

an isolated environment. The authors achieve a factor of 78 reduction in TCB of an open source .NET TCB implementation.

Among the first TrustZone evaluations, Frenzel et al. present the integration of a microkernel system [Här+05] with the TrustZone platform [Fre+10]. The authors make a full virtualization of the microkernel system Nizza possible by using TrustZone. Frenzel et al. were the first to present the idea of using TrustZone as a virtualization technique in embedded systems.

Hussin et al. implement a mobile ticketing system that uses TrustZone technology [Hus+05]. The authors use the secure kernel to store encrypted keys sent by the operator. Pinto at al. explore TrustZone as a way to enable lightweight virtualization for embedded devices [Pin+14]. The authors virtualize two operating systems, Linux and FreeRTOS, with TrustZone, achieving less overhead when compared to other virtualization techniques.

2.4.2.3. Threats to Trusted Execution Environments

ARM TrustZone

In his work [She15], Shen examines the security of the TEE implementation made by Huawei HiSilicon. The vulnerabilities found in this work affect all devices with Huawei HiSilicon System on a Chip (SoC). Shen exploited a vulnerability in a kernel driver that allowed him to acquire kernel privileges in the *Normal World*. Another vulnerability made the execution of the arbitrary code in the *Secure World* possible. The author proves that shellcode can be executed in the *Secure World* by any TA.

Beniamini presents an exploit of the TEE implementation of Qualcomm [Ben16b] that enables him to execute shellcode in the secure OS. The author exploits the vulnerability in the TA of Widevine DRM and launches the buffer overflow attack. The author argues that his attack might have affected a majority of devices with a Qualcomm chipset at the time.

The versioning of the kernel is not relevant in his attack, as the vulnerability it revealed is in the Widevine DRM TA installed on millions of devices. To examine the implications of shellcode execution within the *Secure World*, in his other work Beniamini breaks the Android Full Disk Encryption (FDE) [Ben16a]. He extracts Qualcomm's *KeyMaster Key* out of the memory of the hijacked KeyMaster TA. The *KeyMaster Key* encrypts the actual keys for the FDE. Finally, the author demonstrates the ability to brute-force the PIN of an Android device using his FRE attack.

Zhang et al. in their work [Zha+16] analyze the issue of cache contention between the *Normal World* and the *Secure World*. The *NS* bit indicates the system by which the cache line of two worlds is used. This eliminates the need for cache flashing while switching between the worlds, however, also causes a contention.

The authors present their timing-based cache side-channel attack that exploits the cache contention. The attack was launched in both user space and kernel space in the *Normal World* on the Android platform. In both cases, the authors were able to extract the AES key from the *Secure World*. Zhang et al. claim that this attack has a broad impact on ARM processors with various Android systems, as the attack exploits the TZ cache architecture.

Intel TXT and SGX Technologies

Wojtczuk and Rutkowska analyze the security vulnerabilities of Intel TXT, the forerunner technology of SGX [WR09]. They identify security problems in the design of TXT and implement an attack that bypasses trusted boot process of TXT, which is one of the cornerstones of this technology.

Weichbrodt et al. [Wei+16] introduce a AsyncShock tool that can manipulate the thread scheduling of enclave code execution. The authors successfully exploit two types of synchronization bugs: use-after-free and time-of-check-to-time-of-use (TOCTTOU) bugs.

2.4.3 Interactive Services

Chapter 2.4.3.1 demonstrates virtualization architectures for interactive services. Privacy and security challenges of currently implemented interactive services are analyzed in Chapter 2.4.3.2.

2.4.3.1. Virtualized Interactive Services

Cheng et al. address the problem of non-elasticity and missing universal accessibility for live broadcasting services [Che14]. They provide a Cloud-based media processing platform MediaPaas to enable live broadcasting for mobile devices from the Cloud. The Cloud offloading, in their approach, is provided for both content providers and STB client devices.

The MediaPaas is based on a distributed media processing model that enables media processing tasks' distribution across VMs. The authors split the media processing tasks into pipelines and smaller tasks so that they can be flexibly assigned to the VM.

In their work Howson et al. identify the fact that common broadcast services do not provide an accurate content synchronization from servers that do not share a common clock [How+11]. The authors enable a combined delivery model for HbbTV services, where content for one video scene can be delivered over broadband and broadcast connections simultaneously. In this system, scene components could be rendered synchronously on multiple terminals and delivered over different networks that have different latencies.

Based on an event timeline sent as an additional element, these scene components could be rendered anywhere on a network and perfectly synchronized with broadcast content. They also present a prototype in which they demonstrate the efficiency of their synchronization performance for 3D video delivery, where different views are delivered over different networks.

With regard to the virtual STB architecture presented in Paper 1 in Chapter 1.6, Franke et al. identify a gap that arises when shifting a STB into the Cloud [Fra+14]. The gap is a missing application eco-system that must be built to provide interfaces between platform operators and application developers.

The authors of this paper present new Cloud APIs for third-party applications concerning this concept. These APIs provide interactivity. They enable applications to encode their media output into the video channel, overlay it on the video channel within user's browser or present it on a second screen. This work focuses on HbbTV-enabled STBs [Hyb16] and is based on the work discussed above [Che14].

2.4.3.2. Interactive Service Security

Ghiglieri and Tews investigate [GT14] the privacy drawbacks within the HbbTV standard. This work is based on the German article [Ghi+13] presented by Ghiglieri, Oswald, and Tews at the German BSI national security conference. In their work, Ghiglieri and Tews show that the use of HbbTV offers the end-user not only advantages in interactivity, but also implies extensive data protection problems. With the advent of HbbTV, the back-channel functionality of HbbTV can be misused by broadcasters and third parties as well for the evaluation of user data.

This work explores the behavior of HbbTV applications and detects additional data being transmitted from receiver to sender that goes beyond the data amount required for standard operations. The authors have also detected some additional server requests by packet sniffing before a user starts an HbbTV interactive service. Ghiglieri and Tews conclude that such requests can only be used for an accurate tracking of users' viewing behavior. In some of the broadcasting periods, even the use of tracking scripts has been detected. The authors show in their work that user tracking methods have already been implemented in Germany and Austria.

These methods enable not only broadcasters and third parties but also neighbors to track user TV behavior on WPA2-encrypted WiFi networks. An attacker can identify the Smart TVs in a WiFi network because the MAC address of a sender is transmitted unencrypted in WPA2. Knowing the typical size and length of packets and typical request time intervals, the attacker can track the usage of HbbTV channels. This is based on the fact that no padding is used to mask the real length of a data packet in WPA2, and all HbbTV data is transferred via HTTP.

All HbbTV terminals must be able to playback non-linear content, as required by the HbbTV standard. Michéle and Karpow investigate Smart TV security and introduce an attack in [MK14], in which they exploit a vulnerability in the media detection functionality of a media player. This media player is offered on nearly every Smart TV and STB. The authors show that through a malicious video embedded in HTML code, an attacker acquires full control over the end-device. This attack is completely undetectable by the end-user.

Lee and Kim [Lee+13] present multiple exploitable bugs in Smart TVs. The bugs have been found in an implementation of software for Smart TVs, e.g. in an installer for Smart TV application store. The user can use them to get root rights on the TV, or they can be used by an attacker. A SamyGO project [Sam16b] provides information for legal reverse engineering and research on Samsung TV firmware.

2.5 Summary

This chapter has provided an extensive analysis of Problems 1, 2, 5.1 and 5.2 that refer to SoA technical and scientific works. The remaining problems stated in Chapter 1 refer to the SRSE architecture presented in this thesis and therefore have not been studied in SoA.

This chapter has introduced and analyzed the SoA of both technologies and research in academia that's relevant for this thesis. First, the MSEEs for media streaming on the Web were discussed. It was also pointed out that this thesis will be focusing on the browser as a

particular case of the MSEE, due to its ubiquitous adaptation on devices and corresponding Web standards.

Also two current problems, listed in Chapter 1.2 as Problems 1 and 2, have been specified for browsers and studied in detail. The analysis performed on Problem 1 and its causes reflects the inconsistent browser support across end-devices due to different device capabilities and vendor implementations. The presented study of Problem 2 highlights the insecure nature of browser technologies.

This chapter has also reviewed the following service technologies: (1) streaming, (2) protected streaming and (3) interactive services. With regard to streaming and protected streaming, a detailed overview of currently available streaming mechanisms and protection techniques has been presented. This chapter also presented a comprehensive study of TEEs that are utilized as advanced security mechanisms in the architecture presented in this thesis. With respect to interactive services, an entire landscape of interactive media has been analyzed.

The service technologies (1) MSE, (2) EME and (3) HbbTV were selected by the author to narrow the research scope. This chapter also presented a study of related works concerning fields of research and possible threats.

The chapter has also studied the insecure nature of the services (2) EME and (3) HbbTV and analyzed the vulnerabilities of EME and HbbTV relevant for this thesis, stated in Chapter 1.3 as Problems 5.1 and 5.2, respectively.

The following chapter uses the results presented here to identify the requirements that the architecture designed in Chapter 4 should fulfill.

3 Requirements Analysis for Secure Remote Service Execution

To address the problems stated in Chapters 1.2 and 1.3, the requirements essential for Secure Remote Service Execution (SRSE) will be defined here. These requirements have been derived from discussions in projects from Deutsche Telekom [Deu17] with partners and standardization bodies, as well as from ongoing contributions of the author to the W3C Cloud Browser Task Force (TF) [Clo17]. Based on the requirements as defined here, the functional architecture of SRSE is then presented in Chapter 4.

This chapter will identify the requirements for three components to build up a functional architecture: (1) the SRSE system and two services[1] for SRSE: (2) Secure Streaming (SES) and (3) Secure Interactive Contents (SIC):

- SRSE: the requirements in Chapter 3.1 focus on the core SRSE functionality that enables the shift of MSEE to a remote server in a secured manner.
- Service 1 - SES: the requirements in Chapter 3.2 securely integrate *streaming* and *protected streaming* into SRSE.
- Service 2 - SIC: Chapter 3.3 defines requirements that adapt *interactive service* to SRSE in a secure manner.

Chapter 3.4 provides an analysis of requirements through the Weighted Scoring Model (WeSM) [O'L09], in which requirements are scored according to weighted criteria. Contradictory requirements are addressed in the architecture through the creation of different architectural models.

Chapter 3.5 summarizes the analysis, provides the requirements' priorities, according to which the functional architecture is built up. In the following, the requirements are referred to as R, the problems - as P.

3.1 Secure Remote Service Execution

This chapter identifies the requirements for the core SRSE functionality: the secure shift of the MSEE to a remote machine. They determine the basis for the SRSE architecture presented in Chapter 4.1.

3.1.1 Remote Function Execution

As discussed earlier for P1, modern Web Media Streaming (WMS) end-devices vary widely in their media service execution capabilities, which causes a massive market fragmentation. To process and execute modern MSEEs, e.g. Web browsers, today's end-devices also need a vast

[1] The services have been derived from the contributions of the author to the W3C Cloud Browser TF and prove the adaptability of SRSE to real-world deployments.

amount of hardware resources that might not be available. This all results in limitations on or even the failing of service execution capabilities. To address P1, this thesis defines R1.

R1 Remote function execution - Functionalities that are not supported by an end-device, and are essential for the execution of WMS, must be enabled remotely.

R1 enables the missing end-device functionalities remotely on a service provider's remote machine and provides the foundation for an Remote Service Execution (RSE) architecture. The remote machine takes over the execution of these functionalities, decreasing the amount of execution on the end-device.

3.1.1.1. RSE Design

To identify the exact design method for the RSE, R2 and R3 must be derived.

Zero Function Loss When designing the RSE architecture, it is important to not lose any State-of-the-Art (SoA) functionality for modern end-devices. The shift of WMS functionality to the remote machine, must not be disadvantageous for the overall platform performance.

It must be ensured that the RSE platform will continue to execute all SoA functions: functional losses are not acceptable. The SoA functions of WMS end-devices will be examined in-depth in Chapter 4.1.1.1.

R2 Zero function loss - The zero function loss must be respected: all of the functions that are necessary for the execution of WMS, and are SoA on current end-devices, must be supported.

Fixed Function Distribution For certain WMS platforms, it has been established that some functions cannot be executed by an end-device. Under those conditions, the functions must be distributed between the server and the client in a fixed manner. This implies a non-dynamic function execution, in which the functions assigned to the server cannot be shifted to the end-device at any given point in time.

R3 Fixed function distribution - Functionalities that are not supported by the end-device must be enabled on a remote machine in a fixed manner.

For a fixed distribution of functions, a fixed placement for a function is defined: either the function is executed on the client or the server. This simplifies the overall service architecture, however, also results in a loss of service flexibility.

In a flexible distribution of functions, the system is designed in an abstract way. The placement of functions could be enabled in a flexible manner. The functions could, therefore, be shifted on the fly from the end-device to the server and vice versa. This architecture is beyond the scope of this thesis, but it is briefly explained in Chapter 7.2.

3.1.1.2. Minimal Intervention in Function Execution

To address P4, end-device functions that fully satisfy the demands of a particular WMS platform, must be preserved. These functions must continue to be executed locally.

R4 Minimal intervention in function execution - The principle of a minimal function intervention must be considered when designing the RSE system: these functions must be executed remotely only if they are not available on the end-device or do not satisfy the demands of WMS services.

This would enable the preservation of end-device functions that are already deployed and satisfy service demands. It would also facilitate the re-use of legacy platform systems that rely on these functions.

Such systems might be operator-specific networks, e.g. cable or broadcast, that reduce the bandwidth used by RSE[2]. This would simplify RSE service integration and acceptability.

3.1.2 Secure Hardware-agnostic System

Browser-based media applications executed locally on an end-device require the local execution of Web browsers, corresponding libraries and frameworks on the end-device.

This execution is highly hardware-dependent and relies on underlying platform-specific libraries and Application Programming Interfaces (APIs) that differ from device to device. The more complex this code is, the greater the probability that the platform could become a target for security threats. R5, defined further below, addresses P2.

R5 Secure hardware-agnostic system - Local code execution on an end-device must be reduced, and end-device dependencies and hardware-dependent interfaces should be resolved where possible.

R5 decreases the amount of security threats to local code execution, analyzed in Chapter 2.1.3. R5 is a foundation for a Secure RSE (SRSE) architecture. Fewer hardware dependencies will also facilitate the decoupling of MSEEs from hardware platforms, addressing P1. This requirement is a contradictory requirement to R4.

3.1.3 Security Functions

When shifting functionality to an SRSE component, an end-device's security functions for the execution of content protection techniques cannot be shifted. They must be preserved on an end-device, as content must be decrypted on an end-device. This is also a logical conclusion of R4. To address P3, R6 has been identified.

R6 Preservation of security functions - The security functions executed on an end-device must be preserved.

R6 does not necessarily stipulate a continuous end-to-end chain of content protection. This is only the case if end-devices support source encryption[3]. Otherwise, source encryption will be re-encrypted by the SRSE component: first it will be decrypted; afterward the security techniques supported by the end-device will be applied in order to deliver the content to the end-device securely.

[2]The Cloud MUI in the context of the RSE is individual for each user and therefore requires unicast delivery, i.e. one-to-one delivery, from the RSE server to the RSE client. On the other hand, the broadcast networks enable a broadcast delivery, i.e. one-to-many delivery, which is much more efficient when delivering live events to a high amount of end-users, e.g. football events.

[3]Source encryption is the encryption scheme utilized at the source of media streaming. It is a part of the overall content protection utilized for securing content in WMS.

The remote SRSE component must support the security functionality of both the source and the end-device. However, the content must be decrypted by the SRSE component and then re-encrypted again, which poses additional security risks for the overall content flow.

In some cases, the interruption of the entire end-to-end content delivery chain through re-encryption is unacceptable. For these cases, the end-to-end security of the content delivery system must be preserved, as stated in R7 below.

R7 Preservation of end-to-end content protection - The end-to-end content encryption must be preserved: the content flow protected by source encryption may *only* be decrypted on the end-device.

3.2 Secure Streaming

This chapter identifies the requirements for SES service, addressing P3 and P5. Furthermore, this chapter also defines an advanced security technique for SES, addressing the vulnerabilities in *protected streaming* technology stated in P5.1. The requirements identified in this chapter provide the basis for the SES architecture presented in Chapter 4.2.

3.2.1 Streaming

R8 and R9 address P3 and P5. In following, a distinction will be made between streaming processing and streaming endpoint. Streaming processing refers to the execution of streaming logic, e.g. ABR logic analyzed in 2.2.1. The streaming endpoint is a destination client for the streaming source.

3.2.1.1. Streaming Translation

With regard to end-devices that do not support source streaming formats[4] and source encryption, the secure translation of content streaming must be applied. In other words, the SRSE component must process streaming and content protection mechanisms that are not supported locally by the end-device.

After processing, the SRSE component applies the mechanisms supported by the end-device and delivers the content to the end-device. SRSE performs the transcoding and re-encryption, i.e. *translation*, processes: transcoding to a streaming format and re-encrypting to a content protection technique supported by the end-device.

R8 Secure streaming translation - The source streaming and source encryption techniques that are not supported by an end-device must be translated within the SRSE environment: they must be processed and adapted to the ones available on the end-device.

However, the technology *translation* could only be adapted to a system, for which end-to-end content protection, as stated in R7, does not have to be preserved. This requirement contradicts R7.

[4]Source streaming formats are the streaming formats utilized at the source or origin of media streaming.

3.2.1.2. Distributed Processing

Some devices exist that do support the source streaming format and source encryption mechanisms. However, with regard to MSE and EME, these devices are not able to process these technologies themselves as these are integral parts of the Web browser, because of the novelty of such technology or the non-ability of browser execution.

In such cases, streaming processing must be executed in a distributed way between the SRSE and its end-device. For the streaming source, both the SRSE component and the end-device must act as one streaming endpoint.

R9 Secure distributed streaming processing - The streaming architecture in SRSE must be enabled in a distributed and secured manner without any effect on the streaming source: the functionality of a streaming endpoint must be distributed between an SRSE component and an end-device with support for the source streaming format and source encryption mechanisms.

3.2.2 Secure Streaming

To address P5.1, R10 is specified here. Security architectures with dedicated secure processors and secure libraries have been established over the past decade.

However, as examined in detail in Chapter 2.2.2, the architectures for content protection have undergone significant changes over the last two years. Native integration of content protection mechanisms within browser runtime environments has disrupted previously established security systems and posed serious security concerns.

R10 Protection of device secrets - Device secrets must be protected by tamper-resistant and hardware-assisted execution.

The device secrets essential for content protection must stay protected even when an end-device software is compromised. In other words, these secrets must be stored in an isolated system area. Isolating secrets only and not entire parts of the system would also reduce the code base of the isolated area and minimize its surface exposed to attack. The isolation must be executed in a hardware-assisted way to reach a high level of security.

3.3 Secure Interactive Contents

Addressing P3 and P5, this chapter identifies the requirements for SIC service. This chapter addresses the security vulnerabilities identified in P5.2 by securely designing the SIC. This chapter establishes the foundation for the SIC architecture further specified in Chapter 4.3.

To address P3, R11 is defined; to address P5, R12 and R13 are defined in the following sections.

3.3.1 Secure Integration

In interactive services integrated with SRSE, the traditional end-to-end delivery chain is extended with an SRSE component. This impacts the established security model and increases

the surface vulnerable to attack, as the data is not only processed on a client, but also on an SRSE server.

The SRSE component resides on the premises of the streaming provider and therefore is considered to be trusted. End-devices of the SRSE platform reside on users' premises and are hereby the main attack vector for illegitimate users. R11 addresses P3 and P5.2.

R11 Secure integration - The processing of data related to the interactive service must be secured in SRSE even if an end-device is compromised.

3.3.2 Interoperability

SRSE has a significant impact on local service execution. Frameworks for interactive services, as described in Chapter 2.3, depend on hardware-specific software libraries responsible for local code execution on end-devices. In SRSE platforms, the proximity between MSEE and a hardware platform, with its specific libraries, is not specified.

The techniques described in Chapter 2.3.3 which enable application signaling towards the MUI, could not be executed in a traditional way. A new method for application signaling must be established, as in SRSE, they must be signaled to SRSE and executed remotely.

However, the signaling that must be adapted to the SRSE environment must also be interoperable with the interactive service technologies that currently exist.

R12 SRSE interoperability with traditional interactive services - SRSE must ensure interoperability with interactive services currently supported by end-client devices.

3.3.3 Signaling Processing

Following the description in Chapter 2.3.3, the signaling of interactive applications is a fundamental technique to enable interactive services. Based on previously defined requirements, an adaptation is required for signaling to establish compatibility.

Given that SRSE is a distributed MSEE, service functions might be executed either on a client or a server in a fixed manner, in keeping with R3. Like any other service function, the signaling processing might be either client- or server-sided. As a minimal function intervention is demanded by R4, the following R13 can be identified.

R13 Client-side signaling processing - In SRSE, end-client devices must process the signaling of interactive services in cases where end-devices are capable of such processing.

Devices with support for interactive services enabled must process the signaling locally. However, if end-client devices do not support this functionality, in SRSE, interactive services signaling must be processed by a server. Server-side signaling processing is beyond the scope of this thesis and is presented in detail in Paper 5 in Chapter 1.6; it is also briefly evaluated in Chapter 6.3.

3.4 Analysis of Requirements

The requirements defined previously are analyzed in this section through a weighting process based on the following criteria:

1. **Cross-platform deployability** shows the ability of the service to run on multiple device platforms.
2. **Limited hardware addressed** identifies the extent to which limited, e.g. legacy, hardware is addressed by services despite its limited capabilities.
3. **Value** encapsulates the relative benefit that a requirement provides to any party involved.
4. **Rational resource allocation** describes the optimized resource usage of both server and client, e.g. resource allocation in which an end-device has hardware resources that are not used would be considered rather irrational.
5. **Acceptability** characterizes the relative acceptance of a requirement by any party involved.

For the weighting of requirements the Weighted Scoring Model (WeSM) is used [O'L09], as it provides the possibility to weight criteria, in contrast to the MOSCOW model [S. 97]. In the WeSM the weight of all criteria is 100%, and this weight is distributed between the criteria according to their importance.

In this work, weights are distributed as follows: **Cross-platform deployability** is weighted with 30%, as device fragmentation is the most significant problem addressed in this work; **Limited hardware addressed** is weighted with 25%, as it is also related to device fragmentation, however, this is a subset of the devices dealt with in the first criteria; **Value** has the weight of 20%, as it corresponds to the relative estimation of the importance of requirements; the **Rational resource allocation** is weighted with 15% as optimization is also an important criterion when designing the system and these resources should not be left unused, however, this criterion is less important than those listed previously; the **Acceptability** is weighted with 10% as the acceptance of novel technologies designed in this work is necessary, however, it has the smallest weight as this criterion is just an estimation of how the design proposed in this work might fit into the technology domain of the future.

In order to choose the requirements that are to be weighted, the work of Karl E. Wiegers [Wie99] is used. This work states that weighting must only be applied to requirements that are not logically linked.

The following requirements towards SRSE are considered: R1, R5, R7. Within the service requirements, the following are considered: R8, R9, R10, R11 and R12.

According to the work of Karl E. Wiegers, these requirements are not considered in the weighting: R2, R3 and R4 are logically linked to R1 as further granulations; R6 is linked to R4; R13 is a logical result of R4.

After the criteria are weighted, the requirements are scored on a scale from 0 to 100, according to the extent to which a requirement addresses each criterion. The requirement is scored with 100 if the criterion is fully met and 0 if the criterion is not fulfilled at all, or if it does not relate to the requirement.

Different architectural approaches will address the requirements that contradict each other. These contradicting requirements are weighted in proportion to 100 according to the amount of contradicting requirements, e.g. in the case of two contradicting requirements, each will

be weighted with 50. Contradictions are only considered if the requirements are chosen for weighting.

The weighting of the requirements is presented below in Table 3.1. The table also shows the *weighting reasons* for some of the 100 weightings and all the weightings other than 0 and 100. The final scoring of each requirement is S_{req} and is calculated as stated in Equation (3.1),

$$S_{req} = \sum_{criterion=1}^{5} s_{req} * w_{criterion} \tag{3.1}$$

where the *criterion* refers to Criteria 1 to 5 defined above, the s_{req} is scoring of each requirement for each criterion, and the $w_{criterion}$ is the weight of each criterion, and is calculated as the criterion weight in % divided by 100%.

Tab. 3.1.: Analysis of the Architecture Requirements

Criteria	Criteria Weighting	Secure Remote Service Execution R1 Remote function execution	R5 Secure hardware agnostic system	R7 Preservation of end-to-end content protection	Secure Streaming R8 Secure streaming translation	R9 Secure distributed streaming processing	R10 Protection of device secrets	Secure Interactive Contents R11 Secure integration	R12 SRSE interoperability with traditional interactive services
Cross-platform deployability	30%	80 (It does enable cross-platform compatibly, however the RSE service must be designed separately for each platform.)	100	0	100	50 (Some devices are addressed as the processing is distributed, and the main streaming logic resides on a server, however the media is played back on the end-device, which might not support the newest standards.)	0	0	80 (Through interoperability, all devices that support interactive services are addressed, however the devices that do not support them are not addressed.)
Limited hardware addressed	25%	100	100	20 (The limited amount of hardware that supports source encryption is addressed, however, major legacy devices do not support it.)	100	50 (The same as for this requirement in the *Cross-platform deployability* criterion.)	0	0	50 (The same as for this requirement in the *Cross-platform deployability* criterion.)
Value	20%	100	50 (It contradicts R4.)	50 (It contradicts R6)	50 (It contradicts R7.)	100	100	100	100

Continued on next page

Tab. 3.1.: Analysis of the Architecture Requirements (continued)

		Secure Remote Service Execution			Secure Streaming			Secure Interactive Contents	
Criteria	Criteria Weighting	R1 Remote function execution	R5 Secure hardware agnostic system	R7 Preservation of end-to-end content protection	R8 Secure streaming translation	R9 Secure distributed streaming processing	R10 Protection of device secrets	R11 Secure integration	R12 SRSE interoperability with traditional interactive services
Rational resource allocation	15%	100	50 (Hardware agnosticism does not fully use the hardware resources of the end-device.)	100 (The resources on both service delivery points are used.)	50 (A lot of processing is done on a server.)	100	100 (Secrets can only be stored on the end-device.)	0	100 (Interoperability is required with interactive services executed on end-devices, the end-device hardware is rationally used.)
Acceptability	15%	50 (For privacy reasons, as user data is processed remotely; also content studios and application providers might not accept remote and therefore non-native service execution.)	50 (The same reasons as those given for R1 in the *Acceptability* criterion.)	100	20	50 (The streaming adaptation for SRSE might not be accepted by bodies that standardize streaming technologies, also for privacy reasons, as user data is partly executed on a remote machine.)	100	70 (No security measures currently exist, it is not clear whether the secure integration proposed in this work will be accepted by the standardization bodies or not.)	50 (It is not clear whether the integration of traditional services to SRSE will be accepted by standardization bodies.)
Weight	100%								
Final scoring		89	77,5	40	74,5	67,5	45	27	76,5

3.5 Summary

This chapter presented the requirements that establish the foundation for building the SRSE architecture and the corresponding services SES and SIC. The final scoring of the requirements was presented in Table 3.1, where logically linked requirements were not considered.

The first requirements group for SRSE consists of 3 requirements that have been weighted. According to the final scoring, R1 had the highest score at 89 out of 100, followed by R5 with a score of 77,5 as the second priority and then R7 with the score of 40 as the third priority.

In the second group, the requirements for SES were considered. The highest score was garnered by R8 with a score of 74,5, in second place is R9 with a score of 67,5 and R10 is in third place with a score of 45.

Finally, the third group for SIC was considered. R12 with a score of 76,5 has the highest priority and R11 takes second place with a score of 27.

The following chapter presents the functional architecture based on these requirements. The architecture is built according to the priorities of requirements identified here.

The requirements that contradict each other as identified in Table 3.1 will be considered as the basis for different architectural approaches.

P6 has not been addressed by the requirements as it is a more granular problem that is addressed through the definition of the architecture in Chapter 4.

4 Secure Remote Service Execution Functional Architecture and Specification

This chapter lays out a functional architecture for the novel Secure Remote Service Execution (SRSE) platform and its Secure Streaming (SES) and Secure Interactive Contents (SIC) services. The functional architecture is then extended in this chapter through the detailed specification of the architectural components.

According to this specification, components of SES and SIC services are then practically implemented in Chapter 5. As outlined in Chapter 1.5, the functional architecture addresses the requirements derived in Chapter 3.

The SRSE platform is introduced in Chapter 4.1. This section identifies the core functions that are the essential building blocks of the SRSE platform and builds up the functional architecture for the platform.

The SRSE is specified for a browser-based MSEE and establishes the Secure Cloud Browser (SCB) specification. The SES and SIC services are presented and specified in Chapters 4.2 and 4.3 respectively.

4.1 Secure Remote Service Execution

The requirements defined in Chapter 3.1 will be addressed in this chapter through the shift of the MSEE to the Cloud in a secured manner. Chapter 4.1.1 identifies the basic functionality that serves as a foundation for the functional architecture of SRSE in Chapter 4.1.2. Chapter 4.1.3 provides a detailed specification of SCB, a special case of SRSE that considers a Web browser as MSEE.

According to the priority levels of the requirements stated in Chapter 3.5, they will be addressed as follows: first R1, which is of highest priority, is addressed in Chapter 4.1.1.2 through the definition of SRSE functions, R5, of second priority, is addressed in Chapter 4.1.1.3 by defining the *MUI capturing* function, finally R7 is addressed in Chapter 4.1.3.1 through the Double Stream approach of the SCB.

The non-prioritized requirements are addressed throughout the definition of architecture in the following sections: R2 in Chapter 4.1.1.1, R3 and R6 in Chapter 4.1.2 and R4 in Chapter 4.1.3.1.

4.1.1 Secure Remote Service Execution Functions

This section identifies the functions that form the SRSE architecture. Standard functions of State-of-the-Art Web Media Streaming (SoA WMS) end-devices are identified in Chapter

4.1.1.1. In order to not cause any loss of standard functionality, it is important that these be preserved while designing the SRSE architecture.

As the end-device functionality will be shifted to a remote machine, the traditional service delivery chain is broken up. There will therefore be additional functions that are required by the SRSE architecture. The additional functions caused purely by the shift to the Cloud environment and are not part of the SoA client functionality are identified in Chapter 4.1.1.2. Chapter 4.1.1.3 then identifies the functionality that addresses R5.

4.1.1.1. Media Service Execution Functions

To address R2, the shift of the local service execution to the remote machine must be accomplished without any loss or reduction in the standard functions in WMS services. To satisfy this, the standard functions[1] executed on SoA WMS end-devices are identified for the following groups: application, media, MUI, data, security, control and interaction[2]. These functions enable the foundation for the execution of standard WMS services such as Live TV, Electronic Program Guide (EPG), Video on Demand (VoD).

To execute and enable content streaming services of any kind, an end-device of a WMS platform supports the functions as follows below. These are the required functionalities for the SRSE architecture.

Application Functions - functions that consist of **service execution** and **service composition** functions. The service execution function is responsible for the execution of WMS applications. Such applications are content portal pages, EPGs, VoD portals, etc. The composition function is in charge of a final combination of the MUI elements. The elements are media, graphic elements, e.g. navigations, buttons, teasers, etc.

Media Functions - functions responsible for any media-related operations. They comprise **media rendering**, **media transcoding**, **media decoding** and **channel tuning**. The media rendering function is responsible for a process of creating media frames based on their graphical models.

The media transcoding function converts media from one format to another by applying different media compression techniques. The media decoding function removes the media compression technique previously applied in order to pass on the uncompressed content to the media rendering function.

MUI Functions - functions that are responsible for a final MUI. Here a **MUI rendering** function is considered. It is responsible for the creation of the final MUI by painting the image frames based on their graphical models.

The graphical models are built by the service execution function. As analyzed in Chapter 2.1.2.2 for browser-based MSEE, a Web browser generates a render tree based on the HTML DOM elements. The render tree is then passed over to the graphics context to execute a painting command that renders the final MUI.

[1] Definition of the standard functions is based on the work presented in [Suk+15] and also on the author's contributions to projects of Deutsche Telekom.

[2] Only core functionality is considered here, and such functions as AAA (Authentication, Authorization, and Accounting), as they are secondary and vendor-specific, are not considered.

Data Functions - functions that are responsible for the handling and management of any data available on the end-device. A basic function that will be considered here is a **fetching** function. This function is responsible for downloading the data requested by WMS applications. These data might consist of HTML, CSS, JavaScript (JS) and media data.

Security Functions - functions that are responsible for security-related operations. The security functions provide security on all system layers: application, middleware, platform software and hardware layers. They enable content processing security, watermarking or fingerprinting of content, system security, etc. Here a **decryption** function is considered. It performs content deciphering by applying the required content key of a corresponding DRM or CA system.

Control Functions - functions that are responsible for management and control activities. Control functions depend on a WMS platform. Some platforms execute resource management functions to enable software and hardware resource management for an end-device in multi-user and multi-application platforms [Dej+14].

In this work **management** and **network management** client control functions are considered. The management function is responsible for the creation, maintenance and destruction of sessions. It is also responsible for managing the behavior of the SRSE client by sending the control messages. The network management function enables all network communications.

Interaction Functions - functions that are responsible for interfacing with an end-user in the context of any interactions of the end-user with the platform. This work focuses on an **input processing** function. It receives all types of signals sent by the end-user to the end-device and converts them into the format required by the system.

Other interaction use cases are dependent on a platform, e.g. appliances as second screens, etc. or sensors that could interact with the end-device. However, this work focuses on end-user interactions only.

4.1.1.2. Functions of Remote Execution

To address R1, functions caused solely by the transition to the Cloud environment are defined here. RSE is an additional entity that is placed between the WMS platform back-end and the end-device.

The SRSE must be enabled with functions, usually executed by the back-end, to enable content modifications, manipulations, securing of and further transmission of the content to the end-device. These functions are identified as *Encryptor* functions in Annex B.3.2 and comprise **media encoding** and **encryption** functions. These functions are included in the groups *Media Functions* and *Security Functions* respectively.

The media encoding function is responsible for converting media to different file formats by applying media compression techniques. The encryption function is responsible for ciphering the media content. This function encrypts the content to transmit it over the network to the end-device in a protected manner.

The media encoding and encryption functions must serve as the inverse functions to the device's media decoding and decryption functions respectively, as identified in Chapter 4.1.1.1.

As stated in Equation (4.1), the SRSE component applies the media encryption function *encrypt* to a data set *x*, the data *encrypt(x)* is then transmitted to the SRSE client. The SRSE client applies the media decryption function *decrypt* to the data set *encrypt(x)* that then returns the data set *decrypt(encrypt(x))*.

The data can be decrypted only if the data set *decrypt(encrypt(x))* is equal to the initial data set *x*.

$$decrypt(encrypt(x)) = x \tag{4.1}$$

The function *encrypt* must be the inverse function to the *decrypt* function to satisfy Equation (4.1).

Analog to this, the SRSE component applies the media encoding function *encode* on a data set *x*, the data *encode(x)* is transmitted to the SRSE client. The SRSE client applies the media decoding function *decode* to the data set *encode(x)* that returns the data set *decode(encode(x))*.

The data can be played back only if the data set *decode(encode(x))* is equal to the initial data set *x*, as stated by Equation (4.2):

$$decode(encode(x)) = x \tag{4.2}$$

The function *encode* must be the inverse function to the *decode* function to satisfy the Equation (4.2).

The **input processing** function of the *Interaction Functions* group is also extended by the processing of the control inputs that the SRSE server sends to the SRSE client to control and manage its behaviors.

4.1.1.3. Secure Hardware Agnostic System

The **MUI capturing** function addresses R5, which demands a reduction in locally executed code. The reduction in the code resolves end-client device dependencies and hardware-dependent interfaces.

The MUI capturing function extends the *MUI Functions* and is responsible for the creation of a media stream out of the MUI executed in SRSE. The MUI capturing function takes on the role of a display in the SRSE environment: the function captures the rendered frames out of the memory and produces a media stream out of them, a so-called *MUI media stream*.

This function replaces all functions for the transmission of code to the end-device that would be executed on it locally. The MUI execution is therefore independent of the hardware of the end-device and does not have to be adapted to the device-specific APIs.

The data that must be delivered after applying the MUI capturing function are pure media data, as the MUI is then delivered as a media stream to the client.

4.1.2 SRSE Functional Architecture

This section presents the SRSE functional architecture. To address R3, the architecture presented here is based on a fixed function distribution. This architecture addresses the case

of the maximum amount of function transition from the end-device to the SRSE server. The maximum function transition is the shift of a complete MSEE to the SRSE component. This approach will be referred to as a Single Stream approach and is one of the SRSE approaches also addressed in Chapter 4.1.3.1.

When applying the case of the maximum function transition from the end-device to the SRSE, there still is a minimal function set that must reside on the end-device to establish SRSE functioning.

The functions remaining on the end-device comprise the SRSE Client. In order to address R6 the decryption function must be executed on the end-device. This is a core-client functionality that cannot be executed remotely.

Following this, the function media decoding, executed directly after the decryption function, must also reside on the end-device. Another important function that must remain local is input processing. This function receives end-user input from any source supported by the device platform and the input from the SRSE server. The function must then forward the user input to the SRSE component.

Last but not least, the network management function must be executed locally as this is an initial function that establishes the connection of the end-device to the Internet. Decryption, media decoding, input processing and network managing are, therefore, the SRSE client's functions.

The functional architecture of SRSE presented in Figure 4.1 comprises three main functional blocks: SRSE server functions, SRSE client functions, and services executed by these functions. As mentioned earlier, the design of the SES and SIC services is presented in Chapters 4.2 and 4.3 respectively.

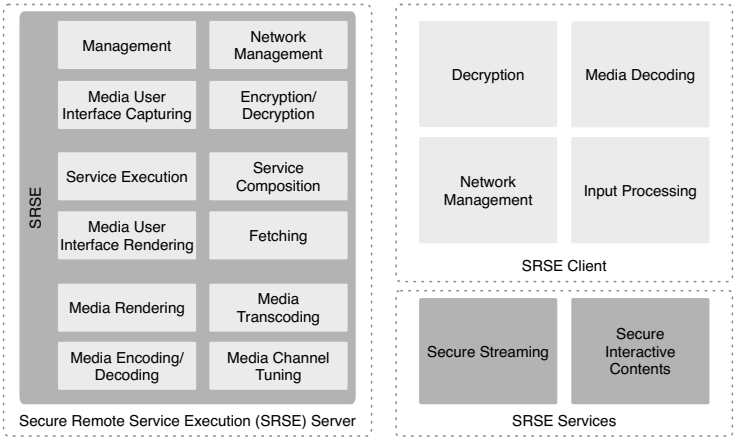


Fig. 4.1.: Secure Remote Service Execution Functional Architecture

This architecture outsources the maximum amount of functions to the Cloud, thus ensuring maximal simplification on the client side. The MUI is delivered to the SRSE client as a media stream. The client performs decryption, if the content is encrypted, and decoding, in order to present the MUI media stream to the display. It also processes the end-user and SRSE server input and enables networking. This is the minimum function set that must be supported by the client.

The functions identified in Chapters 4.1.1.1, 4.1.1.2 and 4.1.1.3, minus the four SRSE client functions, are shifted to the remote machine and comprise the SRSE server. The SRSE server is responsible for the service-related activities like service composition, execution and service data fetching. After the WMS applications have been executed and the MUI has been composed with media from the streaming resources, the SRSE server is responsible for MUI rendering and MUI capturing in an MUI media stream. The graphical operations for the MUI are saved in the buffer, from which the MUI frames are captured and sent to the encoder.

The SRSE server is also responsible for all media-related functions, such as media operations that consist of rendering, transcoding, decoding/encoding and channel tuning, and their security through decryption and encryption functions. The management functions of the SRSE server are session and network management.

The management function is in charge of the sessions, to which the end-devices are assigned, and the client's behavior. The network management coordinates the transport layer, controlling physical links, and handling media channel tuning requests and session handovers.

4.1.3 Specification of Secure Cloud Browser

The functional architecture presented in Chapter 4.1.2 is a general architecture for SRSE that could be realized by various technologies. The SRSE functions responsible for media service execution and composition might be executed by MSEEs currently available and studied in detail in Chapter 2.1.

As discussed previously, this thesis focuses on a standard Web browser as an MSEE for WMS. The specification of a Web browser for the functional architecture defined previously is presented in Figure 4.2 and is referred to as Secure Cloud Browser (SCB).

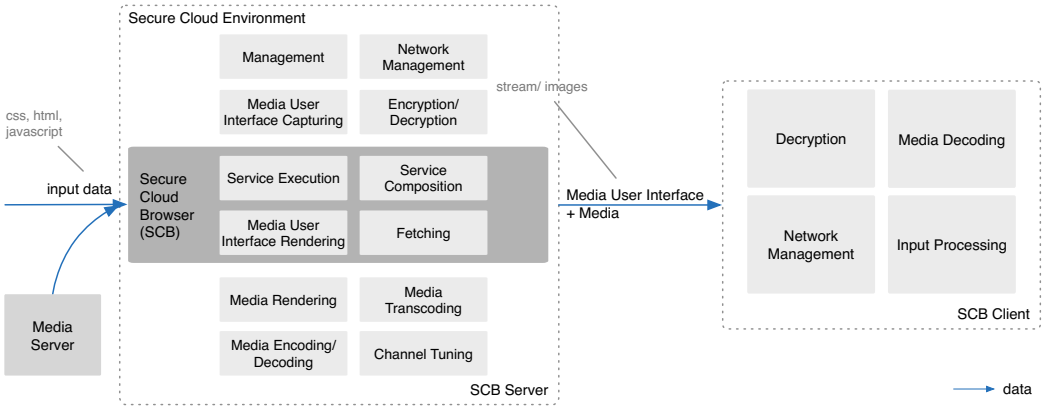


Fig. 4.2.: Secure Cloud Browser Specification

Here, the server functionality is spread over two function executing entities: (1) SCB that acts like any other SoA browser, analyzed in Chapter 2.1.2, and (2) Secure Cloud Environment that manages the SCB.

The Secure Cloud Environment is the platform of the SCB server. SCB executes the following functions: service execution, service composition, MUI rendering, and fetching. These functions are executed by the following browser processes as presented in Table 4.1. These browser processes are analyzed in detail in Chapter 2.1.2.2.

Tab. 4.1.: Secure Cloud Browser (SCB) Functionality Mapping to Secure Remote Service Execution (SRSE)

SRSE Functionality	SCB Process	Process Description
Service Execution	Parsing and code execution	Browser Web core is responsible for parsing the data of a WMS application. After parsing, the JS is passed to the JS core, where it is executed.
Service Composition	Parsing, rendering and embedding of the media	The data is parsed by the HTML parser and CSS parser which build the DOM Tree and the Style Rules, respectively. The JS core applies the changes to the Render Tree. The media is composed with the MUI over the HTML5 `<video>` and `<audio>` tags.
MUI Rendering	Painting and rendering	The DOM Tree and the style rules are a basis for the Render Tree with the rendering layout that is being painted. The painting commands are sent to the platform graphics library over the Graphics Context. The final MUI is then rendered by the graphics library.
Fetching	Fetching	Browser Web core is responsible for loading the application data.

After SCB has composed the final MUI with media by executing these four functions, the final MUI media stream is processed by the Secure Cloud Environment. SCB passes the media data over to the graphics library of the Secure Cloud Environment through the Graphics Context.

The graphics library performs the media-related tasks like rendering, transcoding, encoding/decoding and channel tuning. The media channel tuning function is an abstract function for gaining access to channels.

After the final MUI is composed and rendered, it's also a task of the Secure Cloud Environment to capture the MUI into a media stream, to encode and to encrypt the stream. In case the media is encrypted before it gets to the Secure Cloud Environment, the Secure Cloud Environment must also decrypt it to perform manipulations.

To control the SCB clients, the Secure Cloud Environment also performs management controlling the sessions between the SCB and its clients.

When the SRSE client requests a resource, e.g. a Web page with media elements, the URL is forwarded to the SCB server. The Web browser fetches all the necessary data, i.e. HTML, CSS, JavaScript, media, and renders the page. The rendered page is the final MUI and is captured in a media stream delivered to the client. The final MUI might also be converted, not to a media stream, but to a sequence of pictures in that event that the client does not support low-latency media playback.

The client must only decrypt and decode the media, as the performance-intensive rendering has been already done by the server. Network management also enables the Internet connection for the SCB client.

For communications with the end-user and the SRSE, the client implements an input processing unit that would receive the signals from e.g. the remote control device, translate them into HTTP requests and send them to the SCB; it would also receive control messages from the SRSE and react to them accordingly.

4.1.3.1. Architectural Approaches

The case of the maximum function transition presented above does not completely address the requirements posed in Chapter 3.1. This case of the SRSE is subsequently referred to as the SCB Single Stream approach. In this approach, the SCB captures one single media stream out of the final MUI.

To fully address the posed requirements, the SCB Double Stream approach is considered. This approach does not maximally shift the functions to the SCB; it rather preserves the functions that satisfy WMS demands and covers the case of the medial function transition.

The Double Stream approach addresses R4 and R7. The SCB processes the non-video MUI elements *only*, while the media delivered from another server, therefore called *out-of-band media* [C. 17], is processed by the SRSE client.

Thus, the MUI and media streams are delivered to the client separately, which then has to combine both streams to present them to the end-user in a unified form. The Double Stream approach is examined further in the following.

Double Stream The Double Stream approach is presented in Figure 4.3. The server functionality, analog to the Single Stream approach, is spread over (1) SCB and (2) Secure Cloud Environment.

The functions of SCB remain identical to the Single Stream approach, except for the service composition function. In the Double Stream approach, this function does not integrate any media into the MUI, it only composes MUI elements.

The Secure Cloud Environment does not execute the following functions: encryption/ decryption, media rendering, media decoding, media transcoding and channel tuning. The SCB client is responsible for the following functions: network management, input processing, decryption, media decoding, media rendering, channel tuning and service composition.

The SCB client requests a resource, e.g. a Web application with media elements; the resource URL is then forwarded to the SCB server. SCB establishes the final MUI by executing the WMS application. For this, the application data are interpreted by SCB: it fetches all the necessary data, i.e. HTML, CSS, JS, and renders the page.

SCB then passes the MUI data over to the Secure Cloud Environment, which, in turn, captures the MUI into a stream and encodes it to deliver to the end-device. The media URLs are not fetched by the SCB; they are instead forwarded to the SCB client. The client receives the out-of-band media by fetching the URLs.

The out-of-band media might be delivered over any legacy operator network that is in place. It could also be stored on the end-device and played back locally. The media data is requested by the SCB client directly from the Media Server.

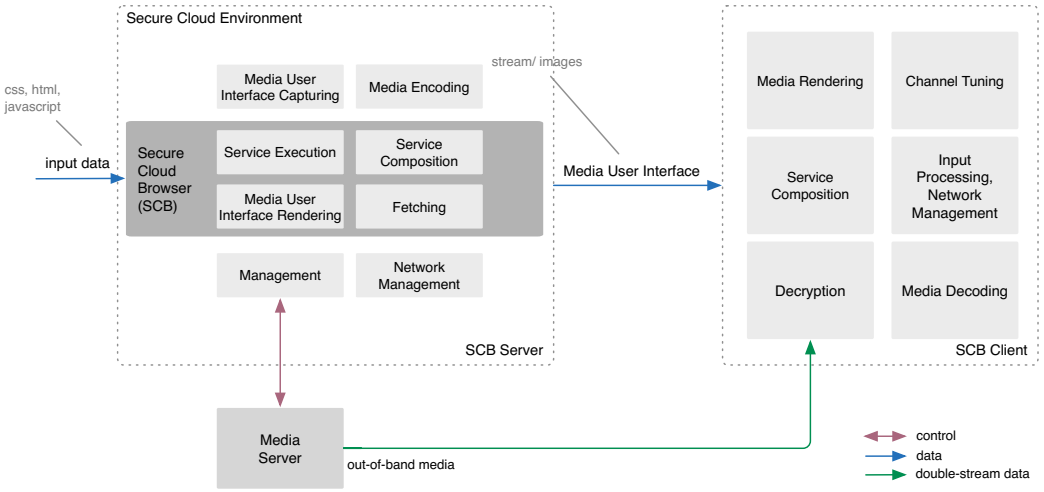


Fig. 4.3.: Double Stream Approach of Secure Cloud Browser

When the content is encrypted, the media data is first processed by the decryption function and then decoded by the media decoding function.

The MUI stream data is received over IP and is processed by the service composition function that composes the MUI stream with out-of-band media. After this processing, the final MUI is played back on the end-device. The tuning of the channels is also conducted locally on the end-device by the channel tuning function.

4.2 Secure Streaming Service

The requirements defined in Chapter 3.2 will be addressed in this chapter through the design of the SES service. Chapter 4.2.1 introduces the functional architecture of SES service.

According to the requirements' relative priority identified in Chapter 3.5, they will be addressed in Chapter 4.2.1 in the following order: first R8 is addressed through the *SRSE Processing* architecture, then R9 is addressed through the *Distributed Processing* architecture and finally the advanced security function addresses R10.

The following chapters provide the specification of the SES architecture for the Web browser: Chapter 4.2.2 specifies the Streaming component of the SES service; Chapter 4.2.3 specifies the Security component of the SES service.

The Streaming and Security components are specified for Media Source Extensions (MSE) and Encrypted Media Extensions (EME) browser technologies. The detailed specification also provides descriptions of interfaces to address P6 stated in Chapter 1.3.

4.2.1 Functional Architecture

In designing the SES service, at first, R8 of highest priority, which demands the secure translation of content streaming, is addressed in this section.

The assumptions made for this requirement in Chapter 3.2.1.1 are as follows: (1) the end-device does not support the streaming format of a streaming source and (2) it does not support the content security technique of the streaming source.

This requirement can only be addressed by the SRSE architecture that addresses the case of maximum function transition. Therefore, the client functions of the Single Stream approach defined in Chapter 4.1.2 should comprise the client functionality and the server functions of this approach comprise the SRSE core in the SES functional architecture.

The SES service is examined in the following. Complying with the assumptions made in R8, the SES service must perform secure content translation between a streaming source and a client.

The content translation is a process of translating content from one streaming format, received from the streaming source, into another supported by the end-device. The secure content translation, i.e. re-encryption, is a process of translating content protection from one technique, applied at the streaming source, into another supported by the client.

The secure content translation process is executed by two SES components. First, the content translation is enabled by the **Streaming** component. This component processes the streaming format received from the streaming source.

In order to translate the media content, the data of this stream is passed to the SRSE Core to be processed by the media transcoding function. This transcodes the former stream into a streaming format supported by the end-device.

Second, the secure content translation is enabled by the **Security** component. This component processes the content protection technique of the streaming source.

This component also coordinates the decryption and encryption functions of the SRSE core. This implies the decryption of the data encrypted by the streaming source stream and its subsequent encryption with a content protection technique supported by the end-device.

Finally, the Security component enables the delivery of the transcoded and re-encrypted media stream to the end-device.

Figure 4.4 demonstrates the functional architecture of the SES service that addresses R8, the so-called *SRSE Processing* architecture. This architecture comprises the SRSE server, SES service and the SRSE client-(a).

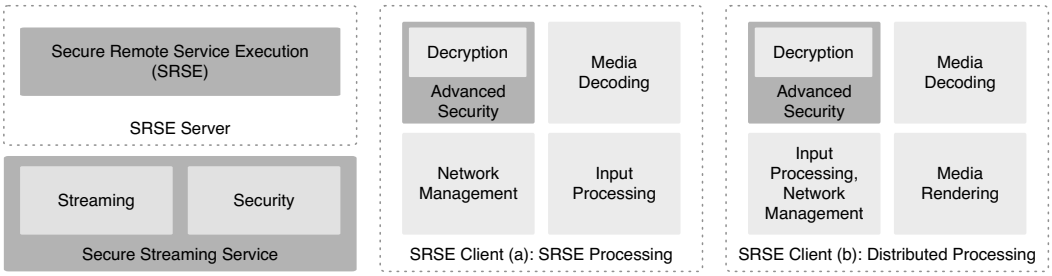


Fig. 4.4.: Secure Streaming Service Functional Architecture: the *SRSE Processing* architecture with SRSE client-(a) and the *Distributed Processing* architecture with the SRSE client-(b)

The functions supported by the SRSE client-(a) are as follows: decryption, media decoding, input processing and network management. As stated in Chapter 4.1.1.2, both decryption and media decoding on the end-device are inverse functions to encryption and encoding

functions on the SRSE server, where encoding is part of media transcoding in the SRSE Core. The Streaming and Security components are the building blocks for the SES service.

R9 of second priority contradicts R8 and is addressed by the functional architecture in Figure 4.4 through the following components: the SRSE server, SES service and the SRSE client-(b). This architecture will be referred to subsequently as *Distributed Processing*.

The R9 meets the assumptions that (1) the end-device supports the streaming format of a streaming source and (2) it supports the content security scheme of a streaming source. Herewith, none of the translation functionality is required.

As specified by R9 the SES processing must be distributed. The Streaming and Security components are therefore executed by both the SRSE server and the SRSE client-(b). To enable the distributed processing, both components are sub-divided into (1) *MSEE-related* processing and (2) *platform-related* processing.

The *MSEE-related* processing processes the SES technology that is part of the MSEE, e.g. corresponding interfaces. The *platform-related* processing processes the SES technology that is part of the device platform, e.g. *format* or *scheme* processing.

The *MSEE-related* processing in both Streaming and Security components is executed by the service execution function in the SRSE core. The *platform-related* processing in both components is executed on the end-device.

The streaming *format* processing in the Streaming component is executed by the media decoding and media rendering functions on the end-device. The content protection *scheme* processing in the Security component is executed by the decryption functions on the end-device.

To enable the media *format* processing in the *Distributed Processing* architecture, the SRSE client-(b) executes the media rendering function as opposed to the tasks executed by the SRSE client-(a).

To address R10, of third priority, the **advanced security** function in both (a) and (b) cases of the SRSE client isolates device secrets. This function is based on the decryption function, as it extends its functionality with additional security mechanisms for content decryption keys.

The Streaming and Security components, the execution of which has generally been identified here, are specified in detail in the following sections. Chapter 4.2.2 addresses the Streaming component; the Security component is specified in Chapter 4.2.3.

4.2.2 Specification of Streaming

This chapter specifies the Streaming component of the SES service for the Web browser and the MSE streaming interface. The *SRSE Processing* model of the functional architecture presented here is specified through the *SCB Processing* approach in Chapter 4.2.2.1. Chapter 4.2.2.2 provides the *Distributed Processing* specification for the *Distributed Processing* model of the SES functional architecture[3].

[3]This specification has been established in a collaborative project executed by Ronen Mizrahi, the founder and CEO of TVersity Inc. [TVe17], and the author of the thesis as part of contributions to the W3C Cloud Browser TF.

4.2.2.1. Secure Cloud Browser Processing

The *SRSE Processing* functional architecture addresses end-devices that do not support streaming formats and encryption of a streaming source. Thus, the Streaming component is specified through the so-called *streaming translation* mechanism between the streaming source and the end-device. The specification is done for SCB and its corresponding W3C interface MSE for ABR streaming. An in-depth study of the MSE interface has been provided in Chapter 2.2.1.2.

The specification that is demonstrated is referred to as *SCB Processing*. The specification presented in Figure 4.5 enables the streaming translation mechanism by processing the ABR streaming in SCB.

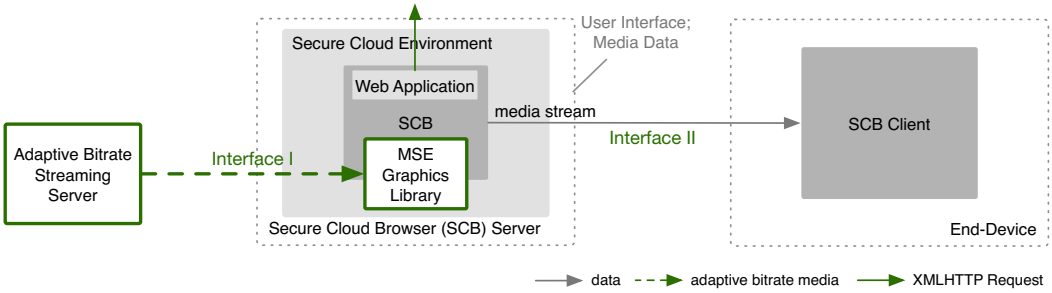


Fig. 4.5.: Secure Cloud Browser Processing of W3C Media Source Extensions

The *Interface I* is responsible for the communication between the SCB and the ABR Streaming Server. The *Interface II* is responsible for the server-client communication between the SCB server and its client. Over the *Interface I*, the SCB initiates a session with the SCB client. The SCB client is executed on an end-device.

The end-device requests the WMS that is implemented through a media Web Application (WebApp), which uses the MSE interface for ABR delivery. The data of the WebApp is classified as HTML, CSS, JS and media data.

The SCB, on behalf of its SCB client, requests the WebApp that contains a pointer to the media data on the ABR Streaming Server. The SCB requests the ABR media stream at the ABR Streaming Server, the server generates the stream upon that request and starts the delivery to the SCB.

ABR content is being delivered to and processed by the MSE Graphics Library component of the SCB. The MSE Graphics Library of the SCB executes the Streaming component. This library enables the processing of ABR in JS, which is the high-level programming language.

The JS processing fills the memory buffers on the SCB server with segments of media data. These, in turn, are rendered by a graphics library of low-level programming language on the server. After the ABR is processed by the MSE Graphics Library, the final media stream is created.

The SCB server applies its media transcoding function to generate, out of an ABR stream, a media stream that complies with the streaming technique supported by the end-device. Over the *Interface II*, the media stream and the MUI are combined into one single stream that

is then delivered to the SCB client. This specification is therefore applicable to the Single Stream SCB approach, as defined in Chapter 4.1.2.

4.2.2.2. Distributed Processing

The SES functional architecture *Distributed Processing* addresses the end-devices that support the streaming format and content protection scheme of a stream's source. However, these devices do not support the processing of the technology itself. This section specifies the *Distributed Processing* for SCB and its W3C interface MSE.

In this specification, the *MSEE-related* processing is the processing of the MSE interface itself. The *platform-related* processing is the streaming format and content protection scheme processing. Thus, the devices addressed do not support the processing of the MSE interface. Figure 4.6 demonstrates the *Distributed Processing* specification.

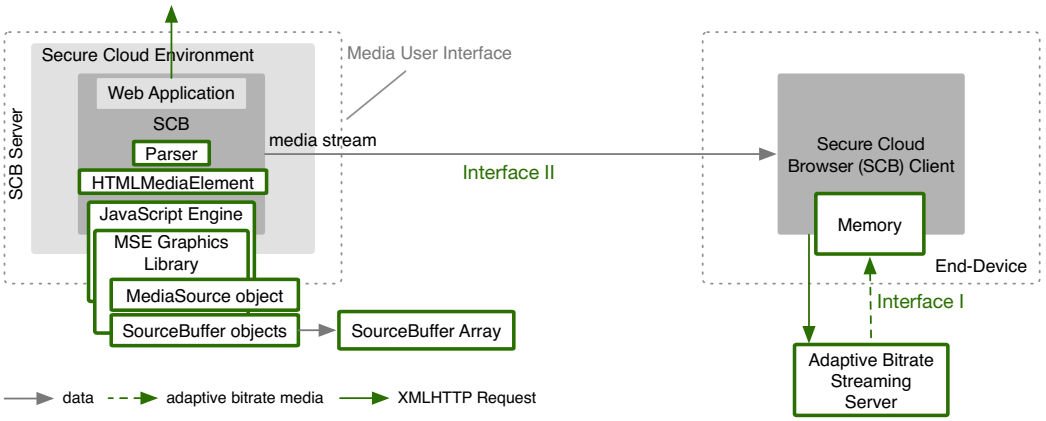


Fig. 4.6.: Distributed Processing of W3C Media Source Extensions by the Secure Cloud Browser

The design of the specification ensures that, for the streaming source, i.e. ABR Streaming Server, the SCB server and the SCB client process MSE as one functional entity, as required by R9. The processing itself is distributed between the SCB and its client. The difference in execution models for the SCB and its client is in the awareness of MSE execution. Here, as the end-device is not capable of the MSE execution, the SCB client is a part of the MSE execution without any awareness of MSE: the SCB client performs the execution on behalf of the SCB.

The SCB is responsible for the execution of the complete MSE logic by the service execution function and forwards only the commands that the client can execute to the SCB client: make a Web request for media, load the media content, render and present the content to the display of the end-device[4].

The SCB logically reconstructs the ABR media stream using the MSE Graphics Library and forwards to the client only the requests that are required to load the data of the stream. The MSE Graphics Library executes the service execution function within the JS engine of the

[4]This execution can be done by the end-device as ABR comprises media formats that are supported by current end-devices.

SCB. The SCB processes the *virtual ABR stream*, while the SCB client processes the *real ABR stream*.

The *Interface I* is the interface over which communication between the end-device and the ABR Streaming Server takes place. The *Interface II* is responsible for the server-client communication between the SCB and its client. By analogy, the initial functional flow of the media WebApp requesting is identical to the *SCB Processing* design.

The further execution flow in this specification is performed as follows. The flow steps are sequentially numbered so that they may be referenced later. **(1)** After the media WebApp is requested by the SCB client, the SCB parses the WebApp-related data being downloaded, and the WebApp sets the HTMLMediaElement. **(2)** The WebApp requests the ABR manifest file and transfers it to the SCB for the execution of parsing. **(3)** The WebApp creates the MediaSource object and associates it with the HTMLMediaElement. MediaSource creates SourceBuffer objects that, in turn, append media segments to the SourceBuffer array through the appenBuffer method.

(4) The WebApp executed in SCB generates the media segment URLs including e.g. the media byte range, video segment ids, etc. **(5)** The WebApp sends the XMLHTTP Requests (XHR) towards the ABR Streaming Server to request these media segments. These requests are forwarded by the SCB to the SCB client so that the client can execute them.

(6) The SCB client loads the media segments on behalf of the SCB and buffers the data in the memory of the end-device. **(7)** Media segments at the SCB server are virtually appended through the appendBuffer method. **(8)** The SCB client decodes, renders and displays the media on the end-device by executing the media decoding and media rendering functions.

The Streaming component is distributed between the service execution functions of the SCB and the media decoding/ media rendering functions of the SCB client. Over the *Interface II*, the SCB delivers the MUI stream to the SCB client. The ABR media is delivered over the *Interface I*. This specification is thereby applicable to the Double Stream SCB approach, as defined in Chapter 4.1.3.1.

Regarding the applicability of the developed *Distributed Processing* specification to real-life deployments, this specification requires execution stages that cannot currently be executed in a standard way according to the W3C MSE standard. The following identifies the logical stages in the execution that diverge from the standard and then presents the solutions proposed in this work. The divergences are numbered consecutively for further evaluation in Chapter 6.2.1.

Stage 1: Execution of XHRs
In step **(5)** in the demonstrated execution flow, the media segments in this design are downloaded by the end-device, the XHRs are therefore forwarded by the SCB to the SCB client. However, the SCB executes a variety of different XHRs; only those that are MSE-related shall be forwarded to the SCB client.

Here Divergence **DIV1** arises: *in MSE the Web browser currently does not have any metadata information about the type of XHRs and therefore does not have any mechanisms to select the required ones.*

To address **DIV1**, the solution referred to as *Additional XHR Type Parameters* establishes additional type parameters of XHR that uniquely identifies the *MSE request* type. Herewith,

the SCB forwards to its client the MSE-related XHRs, which are identified as such by the MSE parameter type.

Stage 2: Loading of media segments
In step **(6)**, the SCB client should download the media segments according to the ordering required in the manifest. However, the manifest is available *only* at the SCB. The WebApp executed by the SCB should send XHRs sequentially towards the Media Server requesting the media segments: the order of the requests is a fixed sequence and should be simply repeated by the SCB client. However, in praxis, the WebApp can load the segments in any order.

With regard to the loading of media segments, Divergence **DIV2** arises: *the ordering of the data is impacted, because the WebApp loads the segments in any order. Therefore, the WebApp, i.e. the appendbuffer method used in step **(7)**, has a different order than the SCB client.* This might be also important when some of the packets have not been downloaded due to bad network conditions.

To address **DIV2**, the *Additional XHR IDs* solution extends XHRs with additional XHR order identifiers that must be created by the WebApp and used by the SCB client as an ordering reference. To become a part of the MSE standard, these identifiers could be included into the XHRs that the SCB would forward to the SCB client. This solution establishes a controllable execution of loading tasks on the SCB client and avoids loading behavior of the end-device that's completely uncontrollable for the SCB.

Stage 3: Buffering while handling the media segments
In step **(7)**, Divergence **DIV3** arises: *there is no way for the SCB client to know which media segment is currently being appended. In other words, the client does not know exactly which segments must be buffered at a certain point in time. The SCB client also cannot differentiate between different resources like audio and video.*

This also has the following implications: if the pre-buffered segments must be removed from the buffer due to changed network conditions, the SCB client also does not know when to clean the buffer, i.e. which segments must be removed.

To address **DIV3**, the following solutions are presented: *Presentation Timestamp Ordering* and *Segment Number Content Type*. The *Presentation Timestamp Ordering* solution reuses the Presentation Timestamp parameter currently used by the appendBuffer method in the MSE standard: this parameter identifies the exact starting time of frame rendering.

This parameter is reused to signal the SCB client the buffering order of the media segments. To become a part of the MSE standard, this parameter could be included into the XHRs that are forwarded to the SCB client. However, should the data be deleted from the middle of a segment, this parameter only indicates the start time and is therefore insufficient.

The *Segment Number Content Type* solution reuses the Segment Number parameter, which is already a part of the XHR URL in the MSE standard. According to this parameter, the sequence of segments can be derived by the SCB client and used as the buffering order.

However, this parameter should be then extended with media type parameters. These would be required to give the SCB client knowledge about content media type, e.g. audio or video. Thus, the SCB client will be able to rebuild audio and video data streams separately in a correct order.

Stage 4: Control
The virtually doubled execution of MSE, where the SCB is aware of the MSE processing, but its client is not, causes Divergence **DIV4**: *the SCB client processes media on its own, which can spin the SCB client out of control as the WebApp does not have any response for the state of current execution tasks on the SCB client. No data is available for the WebApp on the server, the data is physically available on the SCB client only.*

For example, any manipulation of the data as trick functions, e.g. play forward, are not enabled for the WebApp, as none of the data exist on the server. As the MSE JS logic is being executed by the SCB, the data could be appended before it is manipulated. The WebApp might also want to signal to the SCB client only the changes that must be applied to the data.

To address **DIV4**, the following solutions are presented: *Reference Data Return - Media Headers* and *Reference Data Return - Parameters*. In the *Reference Data Return - Media Headers* solution the SCB client returns the media headers to the SCB. This enables the WebApp to perform actions with the data remotely available on the end-device.

In the *Reference Data Return - Parameters* solution, the SCB client returns to the SCB *only* the media headers' parameters required by the WebApp. These parameters are the reference data that are then adjusted by the WebApp and delivered to the SCB client, where the required changes would be applied.

Stage 5: Bitrate Estimations
As the SCB is responsible for MSE processing, MSE-related bitrate estimations are executed by the SCB. Thus, the SCB client might need to send information to the SCB about the available bitrate on the end-device. However, the return of reference data has been proposed in solutions for **DIV4**. Based on these data returned to the SCB from its client, the SCB can calculate the bitrate available on the end-device. Thus, it is not necessary that the WebApp requests the available bitrate from the SCB client.

Distributed Low-level Processing The *Distributed Processing* specification presented above requires multiple adaptations of the W3C MSE standard. Another approach to the distributed execution would be the so-called *Distributed Low-level Processing* model. This model is not within the scope of this thesis and will be briefly used for the evaluation provided in Chapter 6.2.1.

This model enables the separation of the Web browser and therefore MSE functionality through the network, between the SCB and the end-device: the Web browser processes HTML and CSS on the server, the end-device executes the JS engine that is usually a part of the Web browser. Therefore, the end-device also executes the JS MSE logic and the media-related functions, i.e. decoding, rendering. The separating mechanism is enabled by a low-level protocol used for data communication between the SCB and its client.

Over this communication channel, the JS code, after being parsed by the Web browser, is transferred to the end-device for processing within the JS engine. Such a type of MSE processing is not noticeable for the SCB, as from the SCB perspective the complete MSE execution is processed on the server. Thus, the standard MSE processing does not need to be adjusted.

4.2.3 Specification of Secure Streaming

The Security component of the SES service is specified in this chapter for the Web browser and the W3C EME interface for protected streaming. The **advanced security** function is specified through the *Hardware-assisted EME* isolation in Chapter 4.2.3.1.

The *SRSE Processing* model of the presented earlier functional architecture is specified through the *SCB Processing - EME* approach in Chapter 4.2.3.2. The *Distributed Processing* model of the SES functional architecture integrated with the *Hardware-assisted EME* isolation is specified as *Hardware-assisted Distributed Processing - EME* in Chapter 4.2.3.3.

4.2.3.1. Advanced Security

As designed in the functional architecture of the SES service in Chapter 4.2.1, R10 is addressed by the **advanced security** function executed on the end-device. As it has been analyzed in-depth in Chapter 2.2.3, the TEE is within the research scope of this thesis. Thus, the EME processing within a TEE is analyzed here.

Due to the local nature of the execution of this function, it is specified here for the SoA Web browser executed locally on the end-device. Finally, the chosen TEE integration model is extended for adaptation to execution in SCB in Chapter 4.2.3.3.

In the context of the W3C EME interface studied in detail in Chapter 2.2.2.1, the code that is directly involved in the decryption and processing of decrypted media content is executed within the Web browser context in a variety of currently available implementations.

This thesis proposes an isolation of such sensitive code from untrusted applications by transferring the EME execution into the TEE. This isolation is subsequently referred to as *Hardware-assisted EME*.

The main module in EME that integrates the DRM scheme and is therefore responsible for crypto operations is CDM. As analyzed in Chapter 2.1.3, Web browsers are vulnerable to a range of attacks and therefore cannot be considered secure.

Thus, the possibility of executing the complete Web browser with CDM in TEE must be excluded as it would violate the TEE concept. Therefore, at first, the crypto operations are excluded from the Web browser by transferring their execution to the *Platform DRM*, as presented in Figure 4.7.

Platform DRM is a native crypto library on an end-device that is executed by the end-device platform separately from the Web browser. For compatibility purposes with the W3C EME standard, CDM continues to send/receive the calls required by the standard. However, the calls are processed without executing the crypto function itself: CDM functions as a proxy that acts as a mediator between the Platform DRM and the Web browser.

The CDMi model analyzed in Chapter 2.2.2.2, which enables the proxy function, will be used for the purposes of a practical implementation in Chapter 5.2.

The integration of CDM into TEE will be further discussed as the integration of Platform DRM in TEE. The integration of Platform DRM into TEE proposed above might be performed

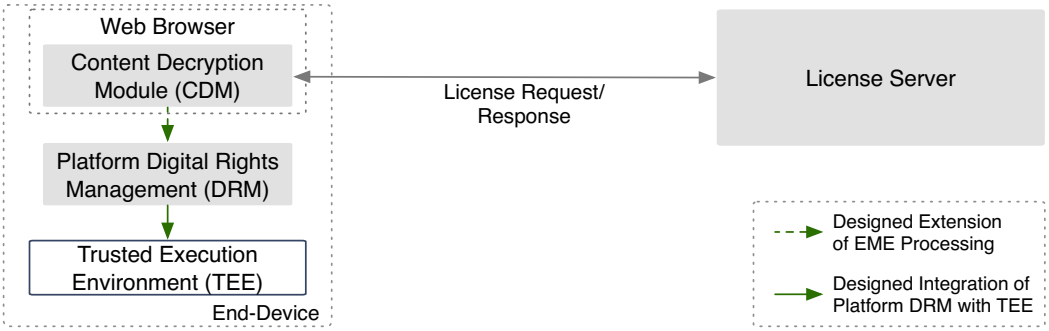
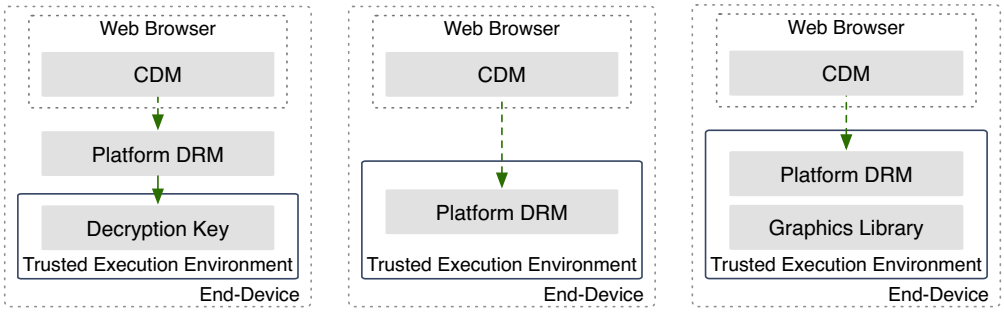


Fig. 4.7.: Specification of Extended W3C Encrypted Media Extensions Processing within the Trusted Execution Environment

to a different extent. The integration approaches reflect varying attack models, as it will be further discussed in Table 4.2.

In the first place, it is important to identify which entities are involved in the processing of DRM-protected content with regard to EME. According to the detailed presentation of EME in Figure 2.3 in Chapter 2.2.2.1, the following parties are processing the content on the end-device: a WebApp that is executed by a Web browser, CDM and graphics library. An additional party would be the Platform DRM, as the crypto functions will be shifted from the CDM to the platform.

Figure 4.8 demonstrates possible integration models for isolating the code that processes protected content inside TEE.



(a) Protection of Decryption Keys in TEE

(b) Protection of Platform DRM in TEE

(c) Protection of Platform DRM and Graphics Library in TEE

Fig. 4.8.: Integration Models of Platform Digital Rights Management into the Trusted Execution Environment

Figure 4.8a presents the *Protecting Decryption Keys* model that only isolates the decryption keys in the TEE. It leaves the decryption mechanism and graphics library that enables media playback outside the TEE. This approach leaves the Platform DRM vulnerable, along with the key exchange process between the TEE and Platform CDM. In addition, the graphics library that processes clear compressed content remains unsecured.

Figure 4.8b demonstrates the *Protecting Platform DRM* model that executes Platform DRM within the TEE. This design protects decryption keys and the whole decryption process,

including usage rights validation. However, once the content has been decrypted, it leaves the TEE and enters the untrusted application zone.

Figure 4.8c presents the *Protection of Platform DRM and Graphics Library* model. This is a complete integration of the Platform DRM and the Graphics Library into the TEE. This model isolates the processing of DRM-protected content from the moment it arrives encrypted from the streaming source until it is securely rendered. This model addresses the vulnerability of the EME, in which the unencrypted media might be passed by the Platform DRM to the Graphics Library in the Web browser context.

Table 4.2 summarizes the approaches presented by also presenting the attack and trust models that have been considered.

Tab. 4.2.: Evaluation of Proposed Models for Integration of the Content Decryption Module (CDM) into the Trusted Execution Environment (TEE)

Characteristics	**Protecting Decryption Keys**	**Protecting Platform DRM**	**Protection of Platform DRM and Graphics Library**
Integration Model	Decryption keys are isolated within the TEE	Platform DRM is isolated within the TEE	Platform DRM and Graphics Library are isolated within the TEE
Attack Model	It is assumed that an attacker has full control over the non-isolated end-device platform, i.e. over the non-secure end-device platform. Due to the high-value of the content, the attacker would like to obtain the cryptographic keys in order to gain access to the content.	It is assumed that an attacker has full control over the non-isolated end-device platform. The attacker might want to reverse engineer the algorithm of a particular DRM library.	It is assumed that an attacker has full control over the non-isolated end-device platform. The attacker might explore vulnerabilities in the Graphics Library of the end-device platform in order to steal and share compressed or uncompressed clear content.
Trust Model	Untrusted parties: Web browser, CDM, platform DRM and graphics library. Trusted Parties: Content decryption keys.	Untrusted parties: browser, CDM and Graphics Library. Trusted Parties: Platform DRM with content decryption keys.	Untrusted parties: Web browser and CDM. Trusted Parties: Platform DRM with content decryption keys, Graphics Library.

As the **advanced security** function protects device secrets *only*, the first model *Protecting Decryption Keys* is considered in this thesis for *Hardware-assisted EME* isolation. Such a model keeps the code complexity within the TEE to a minimum and minimizes its attack surface. The evaluation of the chosen model compared with other models is presented in Chapter 6.2.2.1 after the detailed analysis of its implementation in Chapter 5.2.

4.2.3.2. Secure Cloud Browser Processing

This chapter specifies the Security component in the *SRSE Processing* model of the SES functional architecture, designed in Chapter 4.2.1. The specification addresses end-devices that do not provide support for the content protection schemes from the streaming source.

The component therefore performs a translation of the content protection scheme between the streaming source and the end-device, and is specified for SCB and its corresponding W3C EME interface for protected streaming. The EME interface is analyzed in detail in Chapter

2.2.2.1. This specification develops the translation mechanism by processing the content protection scheme in the SCB and will therefore be referred to as *SCB Processing - EME*.

The foundation for the specification has been demonstrated in Chapter 4.2.2.1 with the Streaming component, specified for MSE. Due to the identical execution logic of specifications for MSE and EME, the *SCB Processing - EME* is briefly presented here.

The interfaces *Interface I* and *Interface II* required for communication are identical to the ones presented in Figure 4.5 of Chapter 4.2.2.1.

The SCB client requests the media WebApp that utilizes the EME interface for playback of protected content. Encrypted media is delivered to the EME component of SCB, analog to the MSE Graphics Library in Figure 4.5. The EME component executes the secure Streaming component.

The SCB processes the EME and re-encrypts the media using the encryption scheme supported by the end-device. To process re-encryption, the decryption and encryption functions of the Secure Cloud Environment are used.

To decrypt the content, the WebApp executed in the SCB requests the keys and licenses of the DRM System specified in the content headers at the License Server, as described in Annex B.3.2. To encrypt the content, the SCB executes the role of Encryptor, specified in Annex B.3.2, and encrypts the content with Content Encryption Keys (CEKs) provided by the License Server of the DRM system that is supported on the end-device.

The integration of EME with TEE analyzed in Chapter 4.2.3.1, might also be considered in this model in order to isolate unique device secrets from each other if multiple end-devices are served by the SCB.

However, in this thesis, the SCB is considered to be a trusted component, as it is executed at the content provider and does not reside at the end-user, and therefore TEE integration is not required. This specification applies to the Single Stream SCB approach, defined in Chapter 4.1.2.

4.2.3.3. Hardware-assisted Distributed Processing

The specification presented in this section further develops the *Distributed Processing* model of the SES functional architecture. This model specified here is for the SCB and its W3C EME interface, which enables the playback of protected content.

The model addresses the devices that support content protection schemes of a streaming source, however, do not support the EME processing itself. The *Hardware-assisted EME* isolation is applied in this specification, as EME processing is distributed between the SCB and the end-device that is located on the user's premises and is considered to be untrusted. The specification developed here will be referred to as the *Hardware-assisted Distributed Processing - EME*.

Regarding the EME, the distributed processing can be applied directly, as the interface is the modular interface by design and consists of multiple entities that forward EME-related messages: the actual crypto functions are only executed by the Platform DRM on the end-device and the License Server on the content provider premises.

The EME interface has been developed with the goal of giving the WebApp control over the execution of crypto-related operations. Therefore, the communication between the License Server and the Platform DRM happens through the WebApp, CDM and Graphics Library.

In the *Hardware-assisted Distributed Processing - EME* specification, the EME processing is distributed between the SCB and its client. The SCB executing the WebApp enables the JS interface EME towards the WebApp. The end-device executes the Platform DRM, Graphics Library, and TEE. The Graphics Library is part of the SCB client and executes media-related functions. The CDM component is executed on the SCB server in a Web browser context. The specification is presented in Figure 4.9.

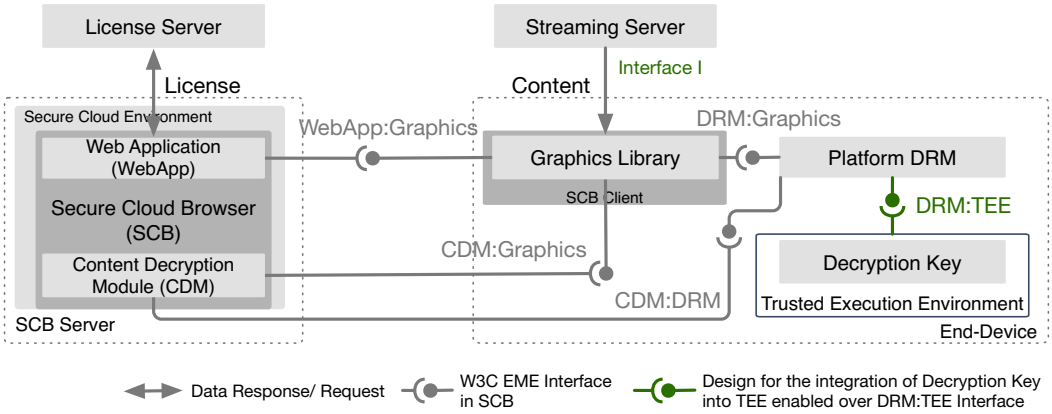


Fig. 4.9.: Hardware-assisted Distributed Processing of W3C Encrypted Media Extensions by the Secure Cloud Browser

The *Interface I* is the interface over which communication between the end-device and the Streaming Server takes place. Any of the streaming technologies studied in Chapter 2.2.1 might be applied here. The server-client communication over the *Interface II* has been already examined in detail in the *Distributed Processing* specification for MSE and is not within the scope of this specification.

The specification consists of the following interfaces: `WebApp:Graphics`, `CDM:Graphics`, `CDM:DRM`, `DRM:Graphics` and `DRM:TEE`. The interface `WebApp:Graphics` is used by the Graphics Library to notify the WebApp that the DRM key is requested. Over this interface, the WebApp also triggers the key exchange session with the Graphics Library. The WebApp requests the DRM key and licenses at the License Server. The media is delivered to the SCB client, i.e. Graphics Library of the end-device.

The interface `CDM:Graphics` is used for communication between the CDM and the Graphics Library. This interface is used by the Graphics Library to trigger the generation of the DRM license request at CDM that, in turn, triggers the Platform DRM. The request itself is generated by the Platform DRM. The content of this request is readable only by the Platform DRM and the License Server. Over this interface, the CDM receives the DRM license and keys from the Graphics Library.

The interface `CDM:DRM` is used by the CDM to forward EME calls to the Platform DRM. The interface `DRM:Graphics` is an interface between the Platform DRM and the Graphics Library of the SCB client for transferring decrypted media samples. The interface `DRM:TEE` is an

interface between the Platform DRM and the TEE, where the CEK is stored. The Platform DRM invokes the TEE over this interface.

The EME interfaces `WebApp:Graphics`, `CDM:Graphics`, `CDM:DRM` and `DRM:Graphics` are defined by the EME standard and are described here as a part of the end-to-end crypto flow, demonstrated earlier in Chapter 2.2.2.1.

The `DRM:TEE` interface is an interface defined in this thesis as part of the proposed SRSE design. It is therefore not part of the EME standard. However, the DRM:TEE interface does not change any data flow within the EME standard and is compatible with current EME implementations. The prototypical implementation of the EME integration with TEE and the corresponding `DRM:TEE` interface will be presented in Chapter 5.2.

The execution flow in this design is performed as follows. The flow is sequentially numbered with the steps that will be referenced in the corresponding implementation in Chapter 5.2. **(1)** The SCB client requests a WebApp that utilizes EME for the playback of encrypted media. **(2)** After the WebApp is requested, the SCB parses the loaded WebApp-related data and the WebApp sets the HTMLMediaElement. Then, the WebApp sets up the MediaKey and MediaKeySession objects.

(3) The SCB client receives the media data from the Streaming Server. The Graphics Library parses the media headers and recognizes that the content is DRM-protected. Over the `WebApp:Graphics` interface, the Graphics Library informs the WebApp that a license is required to playback the content. Over the same interface, the WebApp triggers the Graphics Library to generate the license request. The Graphics Library forwards this trigger request over the `CDM:Graphics` interface to the CDM. The CDM, in turn, forwards this request to the Platform DRM over the `CDM:DRM` interface. **(4)** The Platform DRM generates the key request and forwards it back to the SCB over the `CDM:DRM` interfaces.

(5) The WebApp requests keys at the License Server with an XHR. **(6)** After the keys are received by the WebApp, they are forwarded to the Platform DRM. The keys are encrypted and are forwarded by the Platform DRM to the TEE over the `DRM:TEE` interface. Within the TEE, the keys are decrypted and sent back to the Platform DRM. The Platform DRM applies the keys and decrypts the media data.

Over the `DRM:Graphics` interface the decrypted data is passed to the Graphics Library for decoding and rendering. This specification applies to the Double Stream SCB approach, as defined in Chapter 4.1.3.1.

4.3 Secure Interactive Contents

This section addresses the requirements defined in Chapter 3.3 through the design and further specification of the SIC service.

Chapter 4.3.1 develops SIC functions addressing requirements R11, R12 and R13 according to their priority levels as identified in Chapter 3.5. Chapter 4.3.2 introduces the functional architecture of SIC service.

Chapter 4.3.3 provides the specification of the SIC architecture based on the European HbbTV standard for interactive services. The specification also identifies interfaces to address P6, stated in Chapter 1.3.

4.3.1 Secure Interactive Contents Functions

To design the SIC service, the functions that provide a foundation for its functional architecture are considered in this chapter. R12 of highest priority is addressed in Chapter 4.3.1.1 by defining standard functions that are currently executed by end-devices of interactive service platforms.

R11, of second priority, is addressed in Chapter 4.3.1.2 by definition of the **advanced security** function. R13 is finally addressed in Chapter 4.3.1.3 by introducing the *Client-side Signaling Processing* model.

4.3.1.1. Functions of Interactive Service

The function transfer from local service execution to a remote machine must be performed according to the R12, without a loss of interoperability with currently established interactive services. This thesis focuses on the signaling of interactive applications, as stated in Chapter 1.3.

Thus, to satisfy this requirement, the functions responsible for signaling, currently executed on end-devices of interactive services, are studied and analyzed in detail in this chapter. Preserving the existing functions in SRSE establishes interoperability, as none of the existing functions is devalued, which would otherwise require adaptations of corresponding standards. The definition of functions is based on the analysis of interactive services in Chapter 2.3 and work performed by the author in projects from Deutsche Telekom.

The functionality of end-devices in interactive services includes the standard functions of end-devices of WMS platforms, identified in Chapter 4.1.1.1, and an additional group of so-called *Interactive Signaling Functions* responsible for processing the signaling of interactive services. *Interactive Signaling Functions* are functions that consist of **processing**, **extraction** and **filtering** functions.

The processing function is responsible for the processing of the broadcast signal by the broadcast interface of an end-device. The extraction function is responsible for extracting the signaling data out of the broadcast stream. The data include references to interactive applications.

After extraction, the signaling data are transferred to the next function, i.e. filtering function. This function filters out final *Application Pointers*, which are the references to the application sources. The filtering is executed upon the request of and is sent to the **interactive service execution**, which will be considered further.

The *Interactive Data Functions* and *Interactive Application Functions* groups are beyond the scope of the research and are only considered for completeness. They are also briefly evaluated in Chapter 6.

Interactive Data Functions are functions that consist of **requesting** and **delivery** functions. Both requesting and delivery functions can be executed over the broadcast or broadband interfaces.

After being received, the application data is executed and composed by the **interactive service execution** and **interactive service composition** functions in the *Interactive Application*

Functions group. The *Interactive Application Functions* will be considered in Chapter 4.3.2 as part of the SIC functional architecture. The *Interactive Data Functions* will be considered in Chapter 6.3.3 for evaluation.

4.3.1.2. Secure Integration

R11, of second priority, demands the secure integration of the interactive service into the SRSE landscape. The assumptions behind this requirement treat the end-devices that reside on user premises as untrustworthy and could therefore put the platform at risk if an illegitimate user were to compromise an end-device.

The **advanced security** function addresses this requirement and provides additional security for processing signaling data. This function is placed on the end-device, as the end-device is considered to be an object of attacks.

4.3.1.3. Signaling Processing

The *Interactive Signaling Functions* identified in Chapter 4.3.1.1 might be executed either on the SRSE server or the SRSE client. The execution of these functions on the server will be referred to as the *Server-side Signaling Processing* model. The execution of these functions on the client will be referred to as the *Client-side Signaling Processing* model. To comply with R13 the latter approach will be considered for the design of the functional architecture. Both models are evaluated further in Chapter 6.

4.3.2 Functional Architecture

The *Interactive Signaling Functions* group is the *only* functionality that is explicitly required to be executed on the end-device by the requirements presented for the SIC service in Chapter 3.3. Thus, excluding the *Interactive Signaling Functions*, the case of maximum function transition, as designed in Chapter 4.1.2, is utilized as a foundation for the functional architecture of the SIC service.

Thus, the functions that are executed by the SRSE client comprise the standard SRSE functionality: decryption, media decoding, input processing and network managing functions. The client functions also include the *Interactive Signaling Functions*: processing, extraction and filtering functions.

The server functions correspond to the SRSE server functionality identified in Chapter 4.1.2. Accordingly, **interactive service execution** and **interactive service composition**, briefly mentioned in Chapter 4.3.1.1, are also part of the server execution. The functional architecture of the SIC is demonstrated in Figure 4.10.

The filtering function is responsible for data transfer towards the **interactive service execution** that is part of the SRSE server functionality. In the case of a compromised end-device, the data might be manipulated by an illegitimate user: the data might consist of not only the *Application Pointers* but also pointers to malicious Web pages. Thus, this is the function most critical for security and therefore the **advanced security** function is applied to the filtering function to securely transfer the data to the SRSE server.

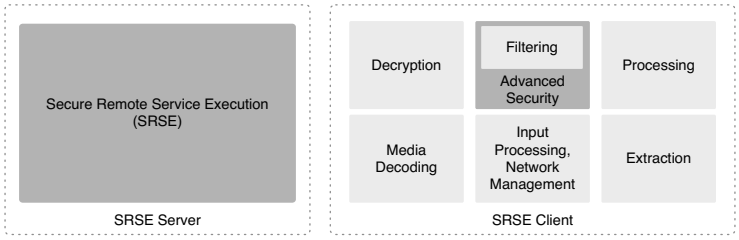


Fig. 4.10.: Secure Interactive Contents Functional Architecture

4.3.3 Specification of Secure Interactive Contents Service

This section introduces the specification of the SIC functional architecture based on the European HbbTV standard, and its functionality. The specification also addresses the HbbTV insecurity analyzed in Chapter 2.3.4.

The SIC functional architecture presented in the previous section is executed by the SRSE server and the SRSE client. The SIC specification is demonstrated in Figure 4.11. The

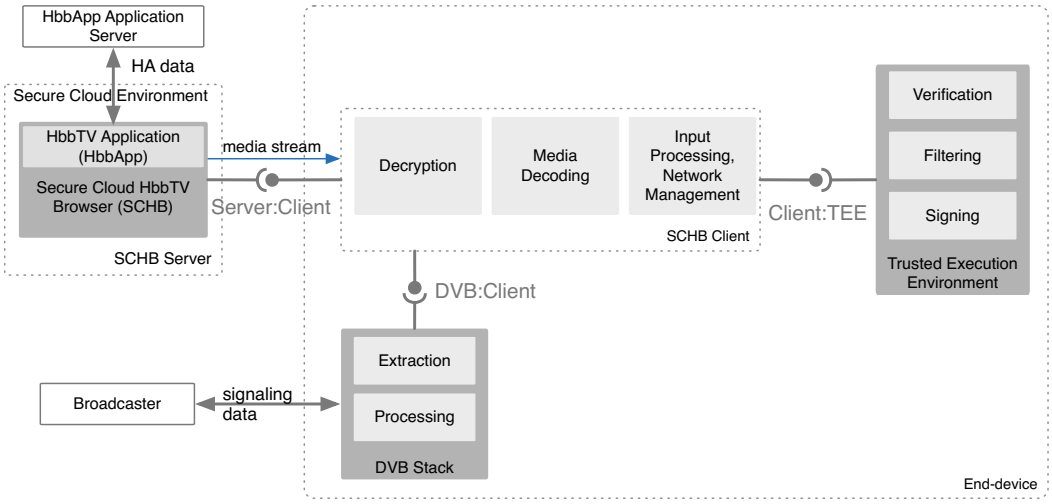


Fig. 4.11.: Specification of Secure Interactive Contents Functional Architecture based on HbbTV and Trusted Execution Environment

functionality executed by the server remains the standard SRSE functionality, excluding the service execution and service composition functions. These functions are replaced with the **interactive service execution** and **interactive service composition**.

In the context of HbbTV, these functions are executed by the so-called HbbTV browser. The HbbTV browser is a standard HbbTV browser that implements the HbbTV-specific JS interfaces of the OIPF Declarative Application Environment (DAE) [Ope16] specification, as analyzed in Annex B.4.1.

The signaling data are the AIT table, which is signaled with the broadcast signal and contains HbbTV Application URLs that are the *Application Pointers*. The application signaling in HbbTV is referred to as Red-button signaling.

The browser also makes use of the HbbTV Application Manager that performs AIT evaluation to control the life-cycle of HbbTV applications. Thus, in HbbTV, the SRSE core functionality

is executed by the Secure Cloud HbbTV Browser (SCHB) on the SCHB server. The functions of the Secure Cloud Environment remain unchanged. The SCHB executes the HbbTV Application (HbbApp), which fetches the application data from the HbbApp Application Server.

The SRSE client functionality is executed by the SCHB client, the DVB stack, and the TEE. The SCHB client executes the standard functions of the SRSE client, as specified for the Single Stream SRSE approach. The HbbTV execution entities are adopted from the analysis performed in Chapter 2.3.3. The DVB stack is responsible for the AIT-related functions and executing the processing and extraction functions.

The DVB stack receives the broadcast signal over the broadcast interface. Over this interface, the broadcast streaming is received as a MPEG-2 TS stream. The DVB stack is implemented by the Demultiplexer and the `oipfWebkit` library. The Demultiplexer processes the MPEG-2 TS stream, selects the AIT data out of the MPEG-2 TS stream, and transfers it to the `oipfWebkit` library. The library extracts and forms the final *HbbTV Application URL* that is the reference to the application source.

The *HbbTV Application URL*, while processed by the `oipfWebkit` library, might contain references to either DVB or HTTP(S) transport protocol descriptor. In the first case the URL is identified with the prefix `dvb://`, in the latter the prefix would be `http(s)://`.

The *HbbTV Application URLs* are contained within the DVB stack and are filtered by the filtering function upon SCHB request. At this stage, when an adversary has access to the *HbbTV Application URLs*, the following attack model is possible. In this case, an adversary is an end-user that has access to the system as a subscriber of an interactive service. He or she intends to launch an attack against the service's back-end. As analyzed in Chapter 2.3.4, the lack of authentication of DVB broadcasters against DVB terminals enables man-in-the-middle attacks by tampering with the *HbbTV Application URL*.

The adversary inserts his/her malicious *HbbTV Application URL* by rewriting the AIT. The malicious URL is forwarded to the SCHB server. As "*by default, broadcast related applications shall be trusted*" in HbbTV [ETS16], the SCHB fetches the malicious application and begins its execution.

If the malicious application is also able to exploit security vulnerabilities in the Web browser engine and bypass the server runtime environment, it might put the whole data center at risk. Herewith, it's not just one single client that's under attack, the entire server environment of the WMS service is at risk.

To address this attack model, the filtering is processed within the **advanced security** function. This thesis proposes the execution of the **advanced security** function by the TEE. The filtering function processed within the TEE is isolated from the execution of the non-secure OS. This provides the integrity of data that is sent from broadcasters to SCHB through end-user devices, ensuring a chain of trust between the broadcaster and the SCHB platform. The HbbTV signaling processing integrated with TEE is referred to as *Hardware-assisted Red-button Signaling Processing*.

Countermeasures against such attacks are currently in processing and will be part of the HbbTV standard in the near future. Based on the official information provided in [Mic15] [Hbb15], it is assumed in this thesis that the AIT: (1) will contain signatures for streams and

certificates with keys to validate those signatures; (2) regular PKI certificates will be issued by a country of the broadcaster. Due to the large amount of data sent by broadcasters within AIT and the computing power needed to sign every application, it is assumed here that (3) the AIT table will be signed in its entirety.

In the specification, the filtering function is thereby extended with the **verification** and **signing** function. The verification and signing functions are part of the trust chain that is established between the broadcaster and the SCHB. According to the *assumption (1)*, the AIT will be signed with the private key of the broadcaster. According to the *assumption (2)*, the public key will be sent in a certificate that is issued by the broadcaster' country.

The verification function within the TEE verifies the signature using the public key of the broadcaster. If the signature is successfully verified, the filtering function filters the *HbbTV Application URLs* requested by the SCHB. To send this URL to the SCHB, the signing function signs the *HbbTV Application URLs* with the private key of the end-device. Thus, the private key of an end-device never leaves the isolated area.

Manipulations of the *HbbTV Application URLs*, after these have been verified and before they are signed, are impossible, as the whole process is executed within the TEE. The signed *HbbTV Application URL* is sent to the SCHB. After verifying the URL, the HbbTV Application Manager makes the URLs available for the HbbTV browser.

The architecture presented here consists of three interfaces: `DVB:Client`, `Client:TEE` and `Server:Client`. The interface `DVB:Client` is used for communication between the SCHB client and the DVB Stack. This interface is used by the client to receive the AIT data.

The interface `Client:TEE` is used by the client to launch the TEE and forward the AIT data and its certificates to it. Over this interface, the verified, filtered and signed *HbbTV Application URLs* are sent back to the SCHB client.

The interface `Server:Client` is used for communication between the SCHB server and the SCHB client. Over this interface the Red-button signaling is realized: verified, filtered and signed *HbbTV Application URLs* are forwarded by the client to the SCHB server upon the SCHB's request.

The execution flow in this design is performed as follows. The flow is sequentially numbered with the steps that are referenced in the corresponding implementation in Chapter 5.3. **(1)** The DVB signal is received by the DVB Stack through the broadcast interface. The signal is processed; the signed AIT data and certificates are extracted from MPEG-2 TS and made available for the SCHB client. **(2)** The SCHB client invokes the TEE and transfers the data to TEE.

(3) The TEE receives the data and verifies the AIT signature, applying the public key of the broadcaster contained in the certificate. **(4)** After the data is verified, the *HbbTV Application URL* requested by the SCHB is filtered out of the AIT table. As broadcast-related applications depend on the broadcasted channel, the filtering of the applications is done based on the channel ID of the broadcast channel that the end-device is currently tuned to. The channel IDs are communicated over the `Server:Client` interface to the SCHB component. The SCHB requests the *HbbTV Application URLs* corresponding to the particular channel ID. The channel

ID, according to which the URLs must be filtered, is sent as a parameter by the SCHB client to the TEE.

(5) The *HbbTV Application URL* is signed using the private key of the end-device. **(6)** The signed *HbbTV Application URL* is passed by the TEE to the SCHB Client. **(7)** The client delivers the URL towards the SCHB over the `Server:Client` interface. **(8)** The SCHB verifies the URL using the public key of the end-device and fetches the application data from the HbbApp Application Server.

The SCHB renders the red button icon into the MUI that is rendered as a media stream and delivered to the end-user. Through the red button icon, the end-user is informed about the availability of the HbbTV application.

4.4 Summary

This chapter has provided the functional architecture and its specification for the SRSE platform. It has also demonstrated how the services SES and SIC fit into and are executed on the proposed SRSE architecture. The architecture addressed the requirements derived earlier in Chapter 3.

This thesis focuses on the browser as an MSEE, and therefore in this chapter, the SRSE has been specified as the SCB platform, where the mapping of the Web browser technology is performed on the SRSE architecture. In correspondence with this, both services have been specified for the SCB. Different architectural approaches have been discussed for the integration of both services into the SCB platform.

In the following chapter, the details for the practical implementation of certain components of the specification will be discussed.

5 Implementation

This chapter provides the practical implementation[1] of the distinct architectural components and corresponding interfaces specified in the previous chapter. Previously, the Secure Cloud Browser (SCB) platform has been specified with the corresponding Secure Streaming (SES) and Secure Interactive Contents (SIC) services executed on the platform. The SCB platform serves as the foundation for the practical implementation, as the services will be executed on top of the platform.

The server and the client components within the SCB platform have been specified in Chapter 4. The server component is beyond the scope of this implementation. Throughout the progression of this thesis and the establishment of the corresponding requirements, industry collaborations have been established with SCB vendors. As a result, several SCB platforms[2] have been either established, elaborated or identified on the market.

During the process of working on this thesis, the existing solutions were analyzed and the *SCB Vendor* solution was selected by the author to use in the test bed as the server component due to its compliance with the requirements derived in Chapter 3.1. Thus, the test bed that is provided here is focused on the SCB client component, client-side SES and SIC service execution and corresponding interfaces.

The implemented system prototype consists of the following components. The core component is the SCB client, the implementation of which will be introduced in Chapter 5.1. Regarding the SES service, the *Protecting Decryption Keys* model of the *Hardware-assisted EME* specification - as specified in Chapter 4.2.3.1 - is implemented for the components residing on the end-device in Chapter 5.2.

Regarding the SIC service, Chapter 5.3 demonstrates the implementation of the SCHB client component with the *Hardware-assisted Red-button Signaling Processing* specified in Chapter 4.3.3.

5.1 Secure Cloud Browser Client

This section presents the practical implementation of the SCB Client. The SCB client is implemented in the Go [Goo17a] and C99 [ISO11] programming languages on the HiSilicon HiKey development board. The choice of the board is explained later in Chapter 5.2.1.

In the implementation the Double Stream SCB model presented in Chapter 4.1.3.1 is developed, as the Single Stream SCB model utilizes only the sub-set of the end-device functionality in the Double Stream SCB model. The Single Stream SCB model can, therefore, be easily executed by the SCB client of the Double Stream SCB model. The details of the

[1] The implementation has been established in a collaboration project executed by the author of this thesis, Robin Stinder and Eugen Osiptschuk [Sti16] [Osi17].

[2] The evaluation and comparative analysis of the SCB solutions are not within the scope of this thesis.

practical implementation are presented in Figure 5.1. The main components that implement the client are the IP Processing, the Control Function, and the Graphics Library.

The client functions specified in Chapter 4.1.3.1 are implemented by the components as follows: the network management function is implemented by the IP Processing; the input processing is implemented by the Control Function; the media decoding, media rendering, and service composition are implemented by the Graphics Library.

Channel tuning is beyond the scope if this implementation. The decryption function will be implemented later by the Platform DRM component in Chapter 5.2.

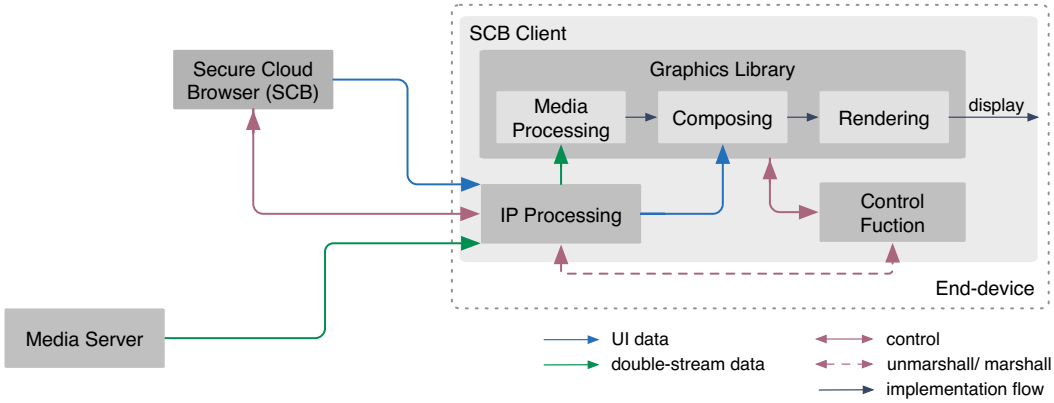


Fig. 5.1.: Implementation of the Secure Cloud Browser (SCB) Client in the Double Stream SCB Model

The IP Processing component processes the TCP/IP stack. The Control Function is in constant communication with the SCB Secure Cloud Environment and the end-user. This component processes the messages received from the SCB server and responds to the end-user inputs, e.g. via a remote control.

The Control Function establishes a session, identified through a *session id*, with the SCB via the SCB Secure Cloud Environment. With the corresponding *session id*, an HTTP communication channel, used as a control channel, is established by the SCB client with the SCB.

Within the Control Function, the messages are unmarshalled when received from the server or the end-user, and subsequently marshalled to be sent to the server. This is done to execute Remote Procedure Calls (RPCs) for data transporting between the processes of different components.

The Control Function processes all incoming messages: depending on message type, it executes different actions. If the server requires a response in a message, the Control Function sends the response. The messages from the SCB server can either query or change the state of the SCB client.

If the SCB server requires the SCB client to open a URL, the control message will be unmarshalled and processed by the Control Function. After processing the Graphics Library is called to playback the URL. The server might send a message controlling the playback of the media or a message that requires the client to stop the playback.

The Control Function component is implemented in Go, and each of the messages received from both the server and the end-user is processed in a so-called *goroutine*[3].

The basic functionality of graphics libraries is the media decoding and media rendering. The media decoding is implemented by the Media Processing component; the media rendering is implemented by the Rendering component. This basic functionality has been extended in this implementation to enable the Graphics Library to compose the media and the MUI streams.

The composition of two separated media streams executes the service composition functions and implements the Double Stream SCB model. The streams from external Media Servers, together with the MUI stream, can be played back as one stream simultaneously: the MUI stream overlays the media stream.

The Media Server is selected by the Control Function that receives the instructions from the SCB server. In the case of the Single Stream, the Graphics Library does not have to execute the service composition function, and it just plays back the MUI stream; the composition is done by the SCB server.

The Graphics Library is implemented in C99 by utilizing the open source graphics library GStreamer [GSt17]. The GStreamer implements programming with data streams that enables transcoding and custom compositions of media from different sources.

The GStreamer pipeline, built to implement the Graphics Library, (1) first injects the data of the MUI stream into the pipeline through the *appsrc* element. Then, (2) the media data is fetched from the Media Server and decoded by the *playbin*.

The *compositor* (3) overlays the MUI and media streams. The *videoconvert* (4) converts the color space to the one supported by the next element (5) *xvimagesink*, which performs the rendering of the final composited stream.

5.2 Hardware-assisted EME

This section demonstrates in detail the implementation of the *Protecting Decryption Keys* model of the *Hardware-assisted EME* and the `DRM:TEE`[4] interface. This implementation is part of the *Hardware-assisted Distributed Processing - EME* specification of the SES service developed in Chapter 4.2.3.3.

As the implementation focuses on the functionality executed on the end-device, the type of Web browser execution, i.e. local or remote, is not of importance here. For the sake of completeness, the Web browser that would normally be executed on the server according to the SCB specification is implemented as a SoA Web browser on the end-device.

All interfaces and components are implemented according to the specification. The interfaces `WebApp:Graphics`, `CDM:Graphics`, `CDM:DRM`, `DRM:Graphics` and `DRM:TEE` are executed locally on the end-device in the demonstrated implementation.

[3] The *goroutine* is a Go approach for processing concurrent events.

[4] The interfaces `WebApp:Graphics`, `CDM:Graphics`, `CDM:DRM`, `DRM:Graphics` are executed as part of the Fraunhofer OCDM/OCDMi open source implementation.

In the real-world deployments of the *Hardware-assisted Distributed Processing - EME* specification, the `WebApp:Graphics`, `CDM:Graphics` and `CDM:DRM` interfaces would be executed in a distributed manner over the Internet, which could be easily made possible due to the modular nature of the interfaces, as stated in the specification.

This chapter presents the implementation of the step **(6)** specified in Chapter 4.2.3.3. The end-to-end specification flow **(1)** - **(5)** is part of the Fraunhofer OCDM/OCDMi open source framework utilized in this implementation.

5.2.1 Implementation Platform

The analysis of available Trusted Execution Environments (TEE) technologies performed in Chapter 2.2.3 has identified the ARM TZ technology as the most deployed technology in embedded and mobile domains. The first and only implementation of the ARM TZ that is both open source and the industry standard is the Linaro OP-TEE, which has been chosen as the TEE technology for the prototypical implementation.

The high-level architecture of Linaro OP-TEE is presented in Figure 5.2a. In addition to the *Normal World* and the *Secure World*, OP-TEE includes three main building blocks: the *optee_client*, the *optee_linuxdriver* and the *optee_os*.

The *optee_client* contains *Exception Level 0 (EL0)* client APIs in the user space of the *Normal World* to enable triggering of the TEE. The *optee_linuxdriver* is the TEE device driver for the Linux kernel, and it calls the TEE in the kernel space of the *Normal World*. The driver is executed in the *EL1/EL2* non-secure OS of the *Normal World*.

The *optee_os* is the secure OS in the *Secure World*, referred to as the *Trusted OS* by Linaro and is executed in *Secure EL1* in kernel space.

The *Secure EL0* Trusted Application (TA) is executed in the user space of the *Secure World*. The *Secure Monitor*, as well as the boot firmware, are executed at the highest privilege level *EL3*.

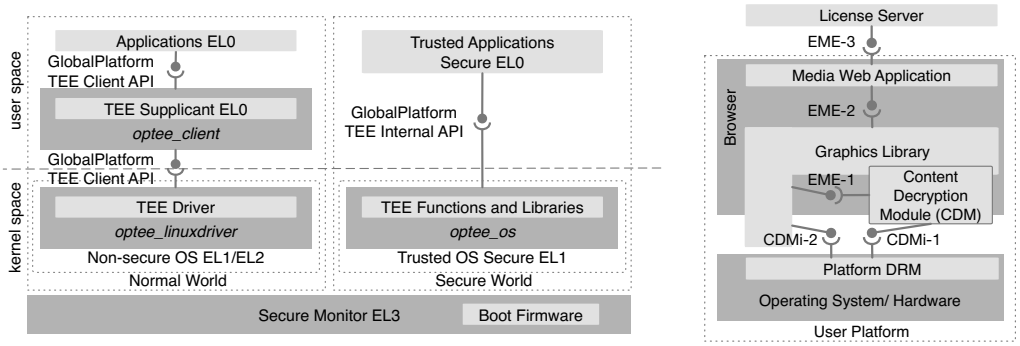
TA is an application that runs in the Trusted OS. TA provides security functions to applications executed in the *Normal World*. TA plays the role of an interface between the *Normal World* and the Trusted OS that runs in the *Secure World*, as specified by the GlobalPlatform [Glo17b].

The security functions are the so-called entry points that, when being called, utilize GlobalPlatform Internal API implemented by the Trusted OS. GlobalPlatform Internal API provides cryptographic and key generation functions for both symmetric and asymmetric algorithms [Sch95]. This API provides the functionality of the Trusted OS to TA, such as a generation of random data required to generate keys, timing or storage APIs.

The hardware platforms that run the OP-TEE are presented by Linaro in [Lin17b]. The preferable platforms are provided by the Linaro 96Boards project [Lin16], which manufactures hardware for developers at a reasonable cost.

The HiSilicon Hikey development board with HiSilicon Kirin 620 SoC and ARM Cortex-A53 CPU has been chosen as a hardware platform for this practical implementation.

To implement the EME interface and remove the CDM processing from the Web browser context, the following technology is used. The open source implementation of the W3C EME



(a) OP-TEE **(b)** OCDMi

Fig. 5.2.: Implementation Platform: Linaro OP-TEE and Fraunhofer FOKUS OCDM/OCDMi

specification and the CDMi [J. 14b] [J. 14a] architecture, presented in Chapter 2.2.2.2, has been developed by Fraunhofer FOKUS Institute.

The W3C EME is implemented by the *Open Content Decryption Module (OCDM)* [Fra14a]. The CDMi is implemented by the *Open Content Decryption Module CDMi (OCDMi)* [Fra14b].

This implementation enables the use of an external *Clear Key* system, as the W3C standard requires the support of *Clear Key* for compliant Web browsers.

Clear Key is a test Web DRM system as it embeds the key into the content header in an unencrypted form, therefore sending it over unprotected. The *Clear Key* system satisfies the demands of this practical implementation, as the *Clear Key* provides sufficient functionality and the crypto functions of DRM systems are beyond the scope of this thesis.

The architecture of the OCDM and the OCDMi is presented in Figure 5.2b. The OCDM is based on the EME [W3C16] specification and is structured as follows:

- **Interface EME-1**: is used for communication between the CDM and the Graphics Library. This interface is used by the Graphics Library to trigger the generation of the DRM license request. The request itself is generated by the CDM.

  The content of this request is understood only by the CDM and the License Server. Over this interface, the CDM receives the DRM license and keys from the Graphics Library.

- **Interface EME-2**: is used by the Graphics Library to notify the Media Web Application that the DRM key has been requested. Over this interface, the Media Web Application also triggers the key exchange session with the Graphics Library.

- **Interface EME-3**: is used by the Media Web Application to require the DRM key and licenses on the License Server.

The OCDMi is based on the CDMi specification [J. 14a] and consists of two interfaces:

- **Interface CDMi-1**: is used by the CDM to forward EME API calls via the CDMi to the Platform DRM. This interface is implemented using RPC.

- **Interface CDMi-2**: is a secured interface between the Platform DRM and the Graphics Library for the transfer of decrypted media samples.

5.2.2 Architecture Implementation

This chapter presents the most important implementation details[5]. Chapter 5.2.2.1 presents the attack model addressed. Chapter 5.2.2.2 presents the implementation of the TA for *Hardware-assisted EME*. This implementation considers the DRM Key Decryption Hierarchy, shown in detail in Annex B.3.2.1. The implemented model provides the isolation of the so-called master key within the TEE. Finally, Chapter 5.2.2.3 describes the integration of EME with the TEE.

5.2.2.1. Attack Model

This section specifies the attack model presented earlier in Table 4.2 of Chapter 4.2.3.1, in the context of the technologies chosen for this practical implementation. In this model, it is assumed that an attacker has full control over the *Normal World* of the end-device platform.

The attacker wants to gain access to high-value content by stealing the master key. The master key decrypts the content keys that are used to encrypt small media segments. With this key, the attacker can decrypt the content keys and illegally copy the DRM-protected content.

The architecture chosen for this work protects the content against such attacks by storing the master key in the *Secure World* of the TEE implemented by ARM TZ. The master key never leaves the isolation of the TEE.

5.2.2.2. Trusted Application

This section presents the implementation of the TA logic for *Hardware-assisted EME* in Figure 5.3. First, the OCDM triggers the OCDMi with a request for decryption of the content. OCDMi invokes the TA logic, as the master key is stored in TEE.

TA logic is comprised of two functional entities, the Host Application (HA) executed in the *NormalWorld*, and the TA executed in the *SecureWorld*.

To call the TA, the HA first calls the TEE Supplicant via the GlobalPlatform Client API. The TEE Supplicant is executed in the user space of the *NormalWorld*. The TEE Supplicant plays the role of mediator between the HA and the TEE Driver.

The TEE Driver is executed in the kernel space of the *NormalWorld* and initiates the *SMC* to launch the TA. Finally, the TA executes the secure code by utilizing the Trusted OS via the GlobalPlatform Internal API.

To launch the TA, the HA first creates a session with the TA. The session receives as a reference an assigned `Universally Unique Identifier (UUID)`. A single TA might handle multiple HAs. Multiple TAs might also be executed within the *Secure World*.

[5]The OP-TEE and OCDM/OCDMi are installed on a 64-bit Debian Jessie Linux OS, provided by Linaro [96B16a]. The OP-TEE used in this implementation is OP-TEE version 1.1.0.

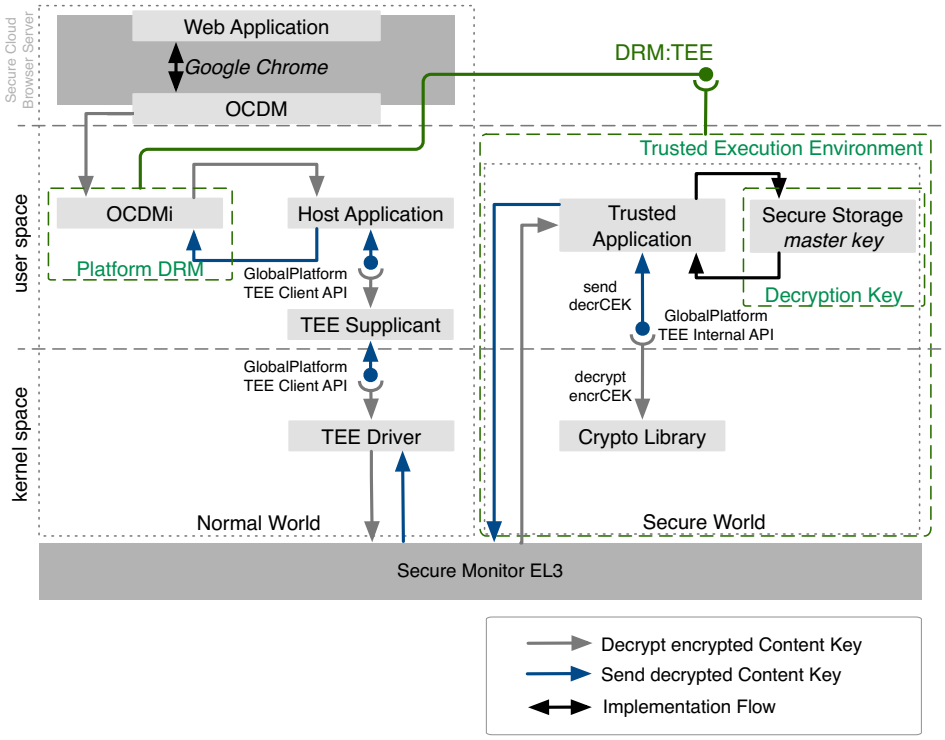


Fig. 5.3.: Implementation of the Trusted Application for the *Protecting Decryption Keys* model of the *Hardware-assisted EME* specification

To create a session the HA makes use of two GlobalPlatform Client API functions: the `TEEC_InitializeContext` and the `TEEC_OpenSession`. The first function uses the `UUID` for the identification of the TA generated in the TA build process.

The session established between the HA and the TA is used to launch and execute the corresponding TA. This is done by the `TEEC_InvokeCommand` function. To initiate the decryption process in the TA, the `TEEC_InvokeCommand` function executes the decryption command.

The encrypted content key is sent by the HA to the TA as an argument of the `TEEC_InvokeCommand` function. The argument is either a memory reference referring to a location in memory where the encrypted content key is stored, or the encrypted content key itself. In this implementation, the memory reference is passed as an argument.

To decrypt the content key, the TA creates a cipher object utilizing the symmetric AES-GCM algorithm. This algorithm was chosen because it is an authenticated encryption that provides both data integrity and confidentiality. Herewith, any manipulations of the encrypted content key can be detected.

The symmetric algorithm is preferred here for its efficiency, as the rotation key model analyzed in Annex B.3.2.1 is applied. In this model, the master key decrypts multiple content keys that in turn decrypt the small media segments. Thus, the model implies a huge decryption effort that will increase when applying the asymmetric algorithms.

Secure storage, as defined in the GlobalPlaform Internal API specification, enables the storage of persistent data of a TA, e.g. keys, inside the *NormalWorld*. The secure storage is implemented in OP-TEE and is called Trusted Storage. The Trusted Storage is beyond

the scope of this practical implementation: in the course of the implementation, the key is generated each time when accessed to provide a simulation of decryption.

5.2.2.3. Integration of EME with TEE

In this practical implementation, EME is extended with CDMi and is implemented through OCDM and OCDMi. The CDMi is part of the architecture proposed in this work, as the actual DRM processing is taken out of the browser context. The OCDM plays the role of a proxy and mirrors the EME commands towards the OCDMi, where the actual Platform DRM resides.

The OCDMi, in turn, communicates with the TA that protects the master key. This communication has already been depicted in Figure 5.3. Figure 5.4 presents the end-to-end system interaction enabled in the practical implementation.

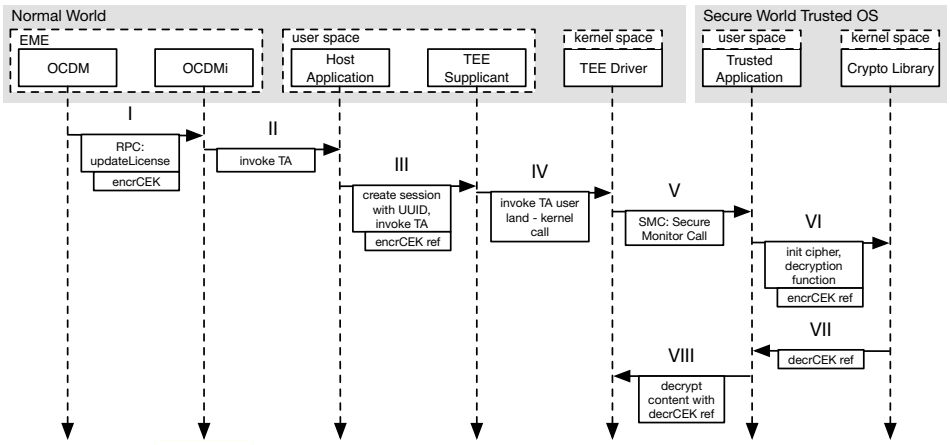


Fig. 5.4.: End-to-end Implementation Flow of the *Hardware-assisted EME*

In this practical implementation, the Clear Key system is used: the encrypted content key *encrCEK* is embedded into the media stream that is parsed by the Graphics Library. The *encrCEK* is then passed to the OCDM over the **Interface EME-1** as described in Chapter 5.2.1.

(I) OCDM triggers the RPC command `updateLicense` containing the license with the encrypted content key *encrCEK*. This command must return the license, with decrypted content key, to decrypt the actual content.

Upon this command (II) the OCDMi performs the call towards the HA to invoke the TA. As a result, the HA creates the session with `UUID` between the HA and the TA via (III, IV) the TEE Supplicant by the *user space - kernel space* call and via (V) the TEE Driver by the *SMC* that invokes the TA.

The calls performed in steps (III) to (V) pass the memory reference *encrCEK ref* of the *encrCEK* to the TA, as the TEE Supplicant stores the encrypted content key in the kernel space of the *Normal World*.

Step (VI) initializes the pre-defined ciphers and the *encrCEK ref* is passed to the Crypto Library for decryption. (VII) The *decrCEK ref* is passed back to the TA, that, in turn, (VIII) sends it to the TEE Driver *Normal World*. The TEE Driver sends the *decrCEK ref* over the

established channel to the OCDMi, which can then start the decryption process of the media.

As discussed in Chapter 4.2.3.3 the interfaces `WebApp:Graphics`, `CDM:Graphics`, `CDM:DRM` and `DRM:Graphics` are part of the standard EME interface. As the signal flow is standardized by EME, a more detailed explanation is beyond the scope of this thesis. The flow is presented in detail in Paper 3.

5.3 Hardware-assisted Red-button Signaling Processing

The implementation of the *Client-side Signaling Processing* model of the *Hardware-assisted Red-button Signaling Processing* specification is demonstrated here. The steps used as a reference in the demonstration of this implementation are identified in Chapter 4.3.3.

To enable step **(1)**, a captured DVB MPEG-2 TS stream is used. In this step, the Control Function of the SCHB client starts the DVBSnoop [Sch16], which implements the functions of the DVB Stack and processes the MPEG-2 TS.

DVB Snoop is an open source, MPEG stream-analyzer program and is utilized in this implementation together with the *translation component* to execute the DVB Stack functionality.

The additional translation component translates the binary AIT format into an XML structure with the MHP syntax [ETS10b]. To translate the AIT information into the XML, the DVB Snoop listens in on the channel-related PID and transfers the received data to the translation component.

In step **(2)**, the SCHB client invokes the TA that is executed in the Trusted OS implemented by the OP-TEE[6]. The SCHB client is implemented by the SCB client in Chapter 5.1. The details of a practical implementation of the TA are depicted in Figure 5.5.

In this implementation, the DVB Stack, after processing the DVB signal, passes the XML AIT table to the HA over the `DVB:Client` interface. The HA in the *NormalWorld* is implemented as part of the SCHB client.

The flow that invokes the TA in the *SecureWorld* corresponds to the triggering of the TA, demonstrated in Chapter 5.2.2.2. The flow that invokes TA implements the `Client:TEE` interface.

The TA verifies the AIT table in step **(3)**, triggering the Crypto Library that utilizes the key out of Secure Storage. For this practical implementation, analog to the implementation of the EME, the required keys are generated each time they are accessed. After a successful verification, the HA invokes the TA to filter the AIT Table to implement the step **(4)**.

In this implementation, the filtering is executed according to the requested Application ID, as the channel ID could only be accessed by the so-called Hybrid Tuner API specified in [OIP14] that is beyond the scope of this thesis. The AIT Filtering function filters the AIT table for Application ID to extract the *HbbTV Application URL*.

[6]The OP-TEE is installed on a 64-bit Debian Jessie Linux OS, provided by Linaro [96B16a]. The OP-TEE used in this implementation is OP-TEE pulled from [OP-16] on 25 May 2017 at 23:10.

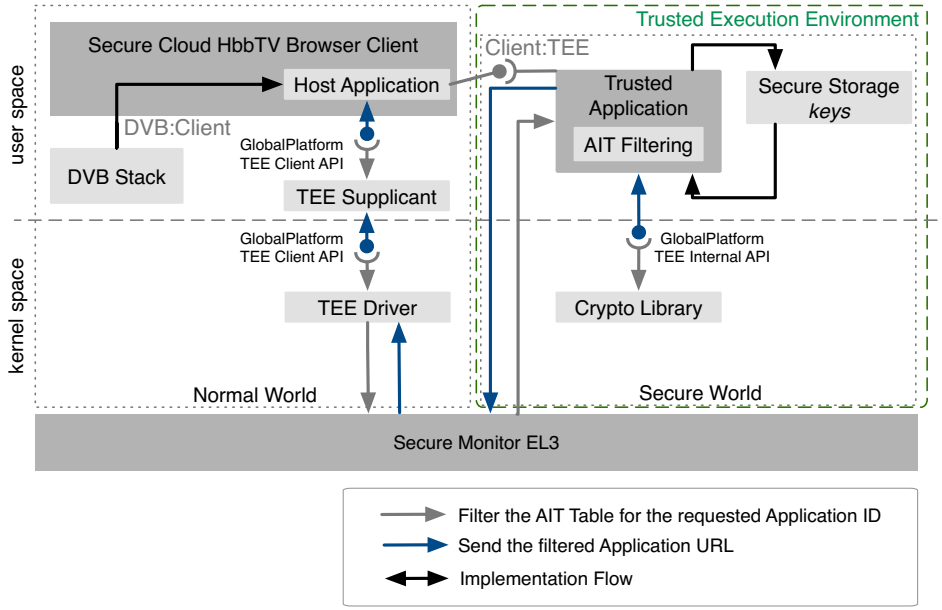


Fig. 5.5.: Implementation of the *Hardware-assisted Red-button Signaling Processing*

The same execution might also be applied to extract other parameters, e.g. application control code, application location, etc. After the data is filtered, to implement step **(5)** it is signed by the Crypto Library utilizing the key out of Secure Storage.

In step **(6)**, the signed data is passed back over the `Client:TEE` interface to the HA in the *Normal World*. In step **(7)**, the data is pushed by the SCHB client to the SCHB over HTTP.

Step **(8)**, which deals with data execution on the server side, is beyond the scope of this implementation.

The Figure 5.6 presents the end-to-end system interaction enabled in the practical implementation. (I - III) Being part of the SCHB Client, the HA invokes the TA and passes to the TA the *data ref*, described previously. The *data ref* are the memory references sent from the HA to the TA and are the references to memory locations of the following data: AIT table, AIT table signature, HbbTV Application ID and also allocated memory references for *HbbTV Application URL* and its signature which will be returned after the TA has been executed in Trusted OS. The *HbbTV Application URL* is also referred to as URL.

(IV) The TA is invoked. The invocation flow corresponds to the flow presented in Figure 5.4. The following presents the specification steps **(3)**-**(7)** in detail.

The TA generates the following keys: the private and public key pair for the AIT table, the private key for the *HbbTV Application URL*. The TA also hashes the AIT table with SHA-256 and signs the hash with the RSA private key for the AIT table. This is done by the TA because no other mechanisms had been provided during the time this thesis was written. In the real world deployment, this would be done by the broadcaster, and the signed AIT table would be sent to the TA. This phase is, therefore, the preparation phase of the TA execution and is not considered for later measurements.

The second phase of the TA is the actual execution phase. Here, three steps are executed. The first step (IVa) verifies the AIT Table. For this first step, the received AIT table is hashed

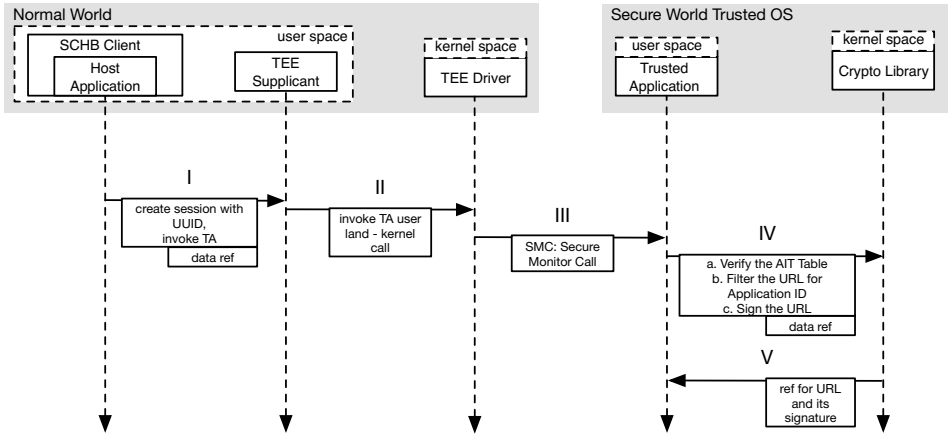


Fig. 5.6.: End-to-end Implementation Flow of the *Hardware-assisted Red-button Signaling Processing*

by the Crypto Library. The hash and the signature of the AIT table are then verified by the Crypto Library with the public key of the AIT table.

Then (IVb) the AIT table is filtered for the Application ID. As a result, the *HbbTV Application URL* is received by the TA. Finally, (IVc) the *HbbTV Application URL* is signed with the private key. For this, the SHA-256 hash algorithm and RSA encryption are used.

(V) The *HbbTV Application URL* and its signature are passed back to the TA, which, in turn, sends it to the TEE Driver of the *Normal World*. The TEE Driver sends the *ref for URL and its signature* over the established channel to the SCHB Client.

The SCHB Client pushes these data to the SCHB. The public key for the *HbbTV Application URL* is available at the SCHB that is used for the URL verification. The SCHB verifies the *HbbTV application URL* and starts its execution in case the verification is successful.

6 Evaluation

In this chapter, a detailed analysis is provided for the evaluation of the designed functional architecture and its specification. The Secure Remote Service Execution (SRSE) and corresponding services will be readdressed with the purpose of analyzing their applicability to real-world deployments. The analysis is conducted on the basis of evaluation criteria identified for each evaluation.

Chapter 6.1 evaluates the SRSE design for Web browser technology and provides discussions of the challenges in the context of such an approach. Chapter 6.2 provides a comparative analysis of the Secure Streaming (SES) specification designs presented in this thesis. It also reports on practical experiments that measure the effect of the overhead of the advanced security caused by the *Hardware-assisted EME* on the overall system performance.

Chapter 6.3 evaluates signaling processing designs and corresponding interfaces for the Secure Interactive Contents (SIC) service. This chapter also provides results of practical experiments to estimate the overhead of the advanced security caused by the *Hardware-assisted execution of Red-Button signaling*.

Appendix B.5 provides the assessment of the related works studied in Chapter 2.4 and presents an assessment methodology for the studied publications in each of the research subjects considered in this thesis. The related works are categorized, assessed according to a defined assessment methodology and compared with the contributions of this thesis.

6.1 Secure Remote Service Execution

This chapter presents the evaluation of the SRSE platform presented in this thesis. First, Chapter 6.1.1 presents a general comparative analysis between the currently established service execution model and the SRSE. Chapter 6.1.2 analyzes the additional network load caused by the introduction of the SRSE platform. Chapter 6.1.3 provides a brief overview of the so-called *Cloud Browser API* that has been established by this research.

6.1.1 Media Service Execution Models

In this thesis, the problems that arise in currently established media service execution models are identified in Chapter 1 as Problems 1 and 2. The research performed in this thesis argues that these problems can be addressed by the shift of local execution into the Cloud and the establishment of SRSE platforms.

The introduction of such platforms raises further questions, stated as Problems 3-6, that have been addressed throughout the research process for this work.

To summarize the most important findings and to compare the SRSE platforms with local service execution, these two models are compared in Table 6.1.

The comparison is conducted based on the following evaluation criteria: the *Application environment* criterion describes which MSEEs, i.e. application environments, are addressed; *Application execution and MUI rendering* shows where the app execution and rendering of the MUI are executed; *Flexible MUI generation* shows whether flexibility while generating MUI remains constant across various end-devices' platforms or not; *Device fragmentation and security issues* analyzes whether the Problems 1 and 2 are addressed; *Integration of legacy end-devices* states if the execution model also addresses legacy devices; *Support of streaming technologies, DRM and HbbTV through a standard approach* shows if the established techniques are addressed through a standard; *Standardization* displays the availability of standards for the particular model; *Network impact* analyzes whether the model impacts the load of existing networks.

The most significant findings will be in-depth analyzed below the table.

Tab. 6.1.: Evaluation of Local Service Execution vs. Secure Remote Service Execution

Evaluation Criteria	**Local Service Execution**	**Secure Remote Service Execution**
Application environment	Is given on the end-device and is not easily interchangeable. It might be a Web app environment, e.g. Web Browser, corresponding interfaces and frameworks, or native app environment.	Multiple app environments are easily interchangeable on the SRSE server.
Application execution and MUI rendering	Client-side	Server-side
Flexible MUI generation	No	Yes, as MUI is generated once on the server-side and could be adjusted at the central location to address all end-device's platforms at any time.
Device fragmentation and security issues	Not addressed	Addressed
Integration of legacy end-devices	Is not addressed	Is addressed
Support of streaming technologies, DRM and HbbTV through a standard approach	The support is given.	Open issue
Standardization	Yes, W3C has established a variety of TV-related Web interfaces.	Open issue, process of Cloud Browser standardization is still on-going in the W3C Cloud Browser TF.
Network impact	No, the existing network load is not impacted.	Yes, because the streams between the SRSE server and corresponding clients introduce a network load that has previously not existed.

The *Local Service Execution* model supports the app environments that are provided on an end-device; these environments are therefore not interchangeable as they depend on the platform of end-devices. This model also struggles with device security and versioning issues on end-devices. It also does not address legacy end-devices that are not capable of supporting the currently established MSEEs.

However, this model provides the already established standard approaches for the integration of streaming, DRM and HbbTV technologies and is based on a variety of already established standards. The Local Service Execution model does not cause additional network load to the one that already exists.

The SRSE model enables the support of multiple app environments as they can be easily interchanged at one central location: the app environment shall only be adjusted once to the hardware of the service machine, none of the end-devices must be regarded. This model addresses the problems of device fragmentation and insufficient security. It also addresses legacy devices, as the computationally intensive execution is shifted to the Cloud.

However, this model does not address traditional and already established technologies, i.e. streaming, DRM and HbbTV, in a standard way. These technologies have been addressed in this thesis through corresponding SES and SIC architectures that have been submitted for standardization within the W3C Cloud Browser TF. However, it remains an open issue if these would become standards.

Finally, one of the main drawbacks that hinders the acceptability of such a model is the network load caused by the newly introduced element - the SRSE server. This network load is a unicast, i.e. one-to-one, traffic that is usually transported by the network established by Internet Service Providers (ISPs).

The advent and usage of CDNs would optimize this delivery. However, CDNs are also using the ISP networks. The model of bandwidth usage on ISP networks will be provided in Chapter 6.1.2.

In summary, the novel SRSE model addresses currently existing problems, while raising new open issues that still need to be addressed. As the past has shown, standards have a huge impact on technology acceptability, and this remains open for SRSE.

6.1.1.1. Execution Time Measurement

The previous section stated that the SRSE platform addresses the device fragmentation issue. This section provides the concrete data related to the differences in browser execution time when different execution models are applied[1].

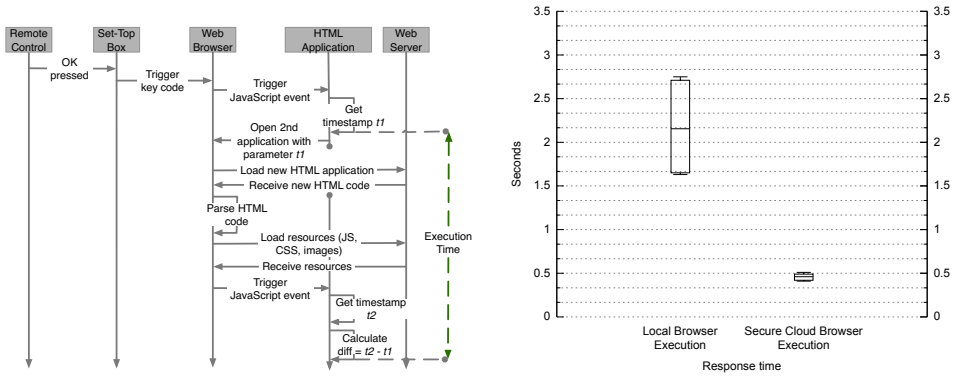
As stated in the motivation statement of this thesis, the device fragmentation problem encompasses all different types of devices. One group of such devices are low-end STBs that struggle with the processing of such a complex MSEE as a Web browser.

To compare service execution speed for the Local Service Execution model with the SRSE model, the following measurement setup is presented. The measurements have been applied to the Local Browser Execution and SCB execution. The entire SCB system, comprising the SCB server[2] and SCB client, that satisfies the requirements posed in this research has been chosen from the *SCB Vendor* and installed on the Linux-based Dune HD TV-102 STB [Dun]. For both execution cases the Webkit-based browser has been utilized.

The message flow of the method proposed for the execution time measurements is presented in Figure 6.1a. The measurement is based on the time stamping performed by two HTML applications. After the remote control command is sent to the STB, the STB triggers the key code and sends it to the browser.

[1] This work was established in a joint project executed by Benjamin Zachey and the author of the thesis.

[2] The SCB server has been implemented on virtualization technology VMware ESXi 5 on a physical host Cisco UCS C200 with 2x 2,4Ghz Intel Xeon E5620 CPUs, 96 GB DDR3–1333-Mhz RAM and 2x 500GB SATA 7200 RPM HDDs in Raid0.



(a) Execution Time Measurement **(b)** Total Browser Execution Time

Fig. 6.1.: Execution Time Measurement Model and Total Browser Execution Time

The browser is located either on the STB itself or the SRSE server, respectively. The connection to the SRSE server is done through the public Internet. The browser triggers the JS event that is communicated to the first HTML application. The application gets a timestamp t_1 and requests the browser to open the second application using the timestamp t_1.

The second application is then requested by the browser with an HTTP request. The second application is a stable HTML5 application that is used as a reference application for the load time measurements. The browser begins to parse HTML code and resources, i.e. JS, CSS, and images, are loaded. Afterward, the browser triggers the second JS event, which sends a second timestamp t_2 to the HTML application.

The time difference t_2 - t_1 is the total browser execution time of the HTML application. The results for the application execution time for the *Local Service Execution* model are presented on the left side in Figure 6.1b. The execution time for the same application applying the SRSE model is presented on the right side in Figure 6.1b.

The average execution time for the first model is 2.175 ms, for the second model is 0.46 ms. The execution time of the SRSE model is 21.15% of the first model. The values are also more stable in the second model and are not broadly spread out as in the first one. This proves that the SCB execution is more constant and faster than the execution on the low-end devices.

It is important to mention that here the entire loading phase of the application data is considered and the measured execution time is related to the entire execution of an application. This, however, does not demonstrate the overall system response latency, as the first application elements are presented to the end-user much faster when compared to the entire execution time of an application.

6.1.2 Network Load Impact

The relocation of the service execution from the end-device to a remote machine in the SRSE model implies an integration of the novel SRSE servers into the currently existing infrastructure of ISP networks.

Independently of whether the ISPs would carry SRSE services from OTT content providers or would provide such services themselves, these services would generate an additional network load.

This chapter presents a model for estimating the required bandwidth consumption by the Single Stream SCB model with regard to the case where ISP providers run SRSE services. This case is examined because this thesis has been executed within German ISP Deutsche Telekom and such a model could be established through multiple prototypes. OTT delivery would also cause additional traffic. However, this case will not be considered in this evaluation.

Concerning the case evaluated, the SRSE servers will be integrated into the existing ISP networks. The networks comprise the following elements to which the SRSE servers might be attached:

- the access network (Outdoor and Indoor Digital Subscriber Line Access Multiplexers (DSLAMs)),
- the aggregation network (Aggregation Switches Level 1 and 2 (AGS1 and AGS2)).

In the access network, DSLAMs are the network elements that play the role of access nodes for end-devices to connect to the network. The aggregation network aggregates all of the traffic coming from the access nodes. Through the aggregation network, the traffic goes into the core of the network, the IP backbone, where the data is processed.

The basic estimation of Live TV scenario bandwidth consumption for the Single Stream implementation was made in a simulation of the scenario with 50.000 ISP customers for the access and the aggregation network.

Every Network Element (NE) allows only for a fixed amount of maximum connected lines, i.e. customers. The maximum number[3] of connected lines per NE and the locations, i.e. number, of the NEs are presented in Table 6.2.

Tab. 6.2.: Bandwidth Consumption Estimation for the Single Stream Secure Cloud Browser Model

Network Element (NE)	**Max. Connected Lines per NE (N_{max})**	**No. of Locations**
Outdoor Digital Subscriber Line Access Multiplexer (DSLAM)	50	600
Indoor DSLAM	100	200
Aggregation Switch Level 1 (AGS1)	1.000	50
AGS Level 2 (AGS2)	10.000	5

The estimation of the bandwidth required is done for the varying ratios R of SRSE service subscribers to all customers. The required bandwidth B_{SRSE} for the SRSE service per subscriber is defined by the Equation (6.1):

$$B_{SRSE} = \sum_{i=1}^{n} B_i \times p_i, \tag{6.1}$$

where the bandwidth B_i with the probability p_i is defined for $n=3$ cases of resource consumption: the usage of HD-channel (9 Mbit/s), the usage of SD-channel (4 Mbit/s) and the concurrent usage of more than one TV-channel - one HD and one SD channel (13 Mbit/s).

[3]The data were acquired from the network departments of Deutsche Telekom and are therefore specific to this ISP.

Therefore, the overall bandwidth that will have to be provided per NE is defined by Equation (6.2):

$$B_{NE} = N_{max} \times R \times B_{SRSE} \tag{6.2}$$

with N_{max} defined in Table 6.2 for three different cases, where the SRSE subscriber ratio R is 0.25, 0.5 or 0.75.

The estimated bandwidth and the number of locations on the network are presented in Figure 6.2. The estimation shows that for a scenario with the ratio *R=0.75* of SRSE subscribers to all ISP customers, the HD-channel usage probability p_1*=0.8*, the SD-channel usage probability p_2*=0.18* and the concurrent usage probability p_3*=0.02* the overall required bandwidth 102.25 Mbit/s on the Indoor DSLAM goes up to 20.45 Gbit/s on the AGS1 with a bandwidth of *B=8.18 Mbit/s* per subscriber.

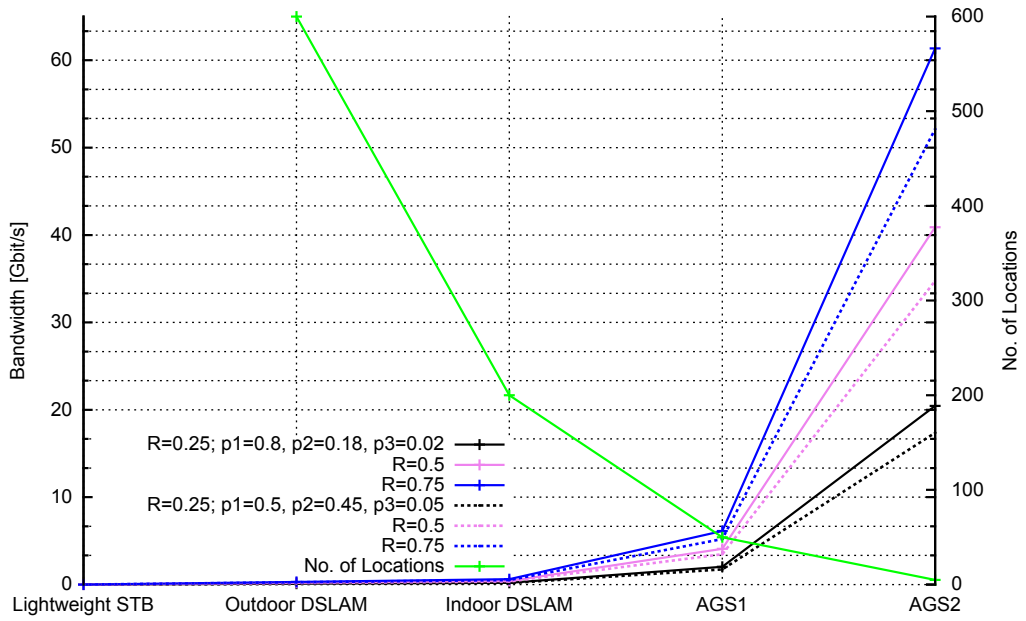


Fig. 6.2.: Bandwidth Consumption Model for the Single Stream SRSE Approach

Due to the change in the channel usage of p_1 from *0.8* to *0.5*, p_2 from *0.18* to *0.45*, p_3 from *0.02* to *0.05* the overall required bandwidth on the Indoor DSLAM is 86.875 Mbit/s and 17.375 Gbit/s on the AGS1 with a bandwidth of *B=6.95 Mbit/s* per subscriber.

Keeping in mind that, currently, a huge part of the media traffic of Deutsche Telekom is transported over the multicast networks, the estimated bandwidth that should additionally be made available on the currently existing networks is quite impressive.

An important issue for Single Stream implementations is the network location trade-off between the distribution of SRSE servers over multiple locations near the end-users with lower bandwidth consumption per NE in the access network, and a few centralized data centers in the metro network or the IP backbone with a significantly higher bandwidth consumption per NE.

The closer the service is located to the end-user, the better the quality of experience provided, however, such an integration wouldn't be very cost-effective due to the number of locations where the physical installations would be taking place. As can be seen on the graph in

Figure 6.2, the AGS1 might be a good placement of the SRSE servers to address the trade-off mentioned.

Considering the significant network load introduced by the Single Stream SCB, the Double Stream approach has been designed in this research to enable delivery of the media content separately through already established multicast and broadcast, i.e. one-to-many, networks.

By reusing the multicast network for Live TV delivery, this approach cuts down on the unicast traffic, which will *only* be used for VOD and MUI stream delivery. In this case, the Live TV multicast traffic will not be incorporated and therefore reduce the load on the unicast network.

The Double Stream approach also requires less server power on the SRSE server, which makes the SRSE infrastructure more cost-effective. However, this approach still demands media processing on the end-device, which partly raises device fragmentation and security issues again.

In the Double Stream approach, where the SRSE client executes the service composition function by overlaying the media stream being played back with the MUI stream, it is important that the MUI stream is made transparent. This ensures that the video can still be seen through other MUI elements, which improves the user experience.

The transparency of the video is enabled through an additional alpha channel. However, the currently widely deployed video codec H.264 does not provide any support for the alpha channel. The new end-devices that will appear on the market in the near future will provide support for High Efficiency Video Coding (HEVC), i.e. H.265, which enables the alpha channel property.

However, millions of deployed end-devices still support H.264, and they need to be addressed. Custom SCB solutions to address this issue that are available on the market deploy a solution that encodes MUI as a stream of images in a PNG format that supports the alpha channel. A disadvantage of such a solution is missing hardware decoders for these image formats as usually provided for H.264.

To address this issue, an alternative approach could be new MUI concepts for the SRSE that eliminate the need for transparency to eliminate the need for the alpha channel. Herewith, widely deployed hardware codecs like H.264 could be used for the MUI stream.

6.1.3 Cloud Browser API

The emergence of the SCB technology exposes a gap in communication between the SCB and the SCB client. A standard interface through which the SCB could communicate with its client does not exist. Multiple solutions are present on the market. However, these are proprietary implementations that deploy vendor-specific communications.

This work has contributed to the standardization of an interface called *Cloud Browser API*. The *Cloud Browser API* is currently under development in the W3C Web and TV Working Group as part of a dedicated Cloud Browser Task Force. This group is working on the unification and standardization of the *Cloud Browser API* and has been established by the author throughout the process of working on this thesis.

6.2 Secure Streaming

In this section two components, namely the Streaming and Security components, designed in the context of the SES service, will be evaluated in Chapters 6.2.1 and 6.2.2 respectively.

6.2.1 Streaming

Three specification models of the Streaming component in SES service are proposed in Chapter 4.2.2: SCB Processing, Distributed Processing and Distributed Low-level Processing. This chapter presents the evaluation of these models in Table 6.3.

First, the *Design summary* is presented, which briefly highlights the main design principles. Afterward, the three approaches are evaluated on the basis of the following criteria: *Requirements addressed:* which of the requirements defined in Chapter 3 are addressed; *Processing entities:* which entities are involved in the processing or execution of MSE; *Awareness of MSE:* which of the design processing entities are aware of MSE processing through the support of the MSE runtime environment; *Processing logic:* which kind of MSE processing logic is presented - the logic is centralized if MSE processing is done by one single entity, and the logic is distributed if processing is performed by multiple entities; *Divergences:* if applicable, which divergences arise with the currently defined MSE standard that need to be addressed in order to enable the model execution; *Required changes in MSE standard:* whether the current MSE standard must be changed or not.

The most significant model advantages and disadvantages will be stated below the table.

Tab. 6.3.: Evaluation of Specification Models of the Streaming Component in Secure Streaming Service

Evaluation Criteria	**Secure Cloud Browser Processing**	**Distributed Processing**	**Distributed Low-level Processing**
Design Summary	ABR is completely processed by the SCB. The SCB transcodes the stream into the streaming format supported by the SCB client.	The processing of MSE is doubled: the WebApp within the SCB context processes MSE, the client loads the data on behalf of the SCB, repeating the SCB requests.	ABR is processed by both the SCB and its client. The distributed processing is enabled by a low-level protocol in such a way that the SCB is not aware that the JS is processed remotely.
Requirements addressed	R8	R9	R9
MSE processing entities	Web Application (WebApp), SCB	WebApp, SCB, SCB client	WebApp, SCB, SCB client
Awareness of MSE	WebApp, SCB	WebApp, SCB	WebApp, SCB, SCB client
Processing logic	Centralized: the WebApp running in the SCB context is a centralized MSE processing entity that communicates with back-end systems, e.g. application back-end or CDN.	Distributed: SCB processes MSE, SCB client also processes MSE without awareness of MSE processing.	Distributed: the SCB processes HTML and CSS data, the client processes JS data.
Divergences	None	**DIV1**, **DIV2**, **DIV3**, **DIV4**	None
Required changes in MSE standard	None	Yes, to address the **DIV1**, **DIV2**, **DIV3**, **DIV4**	None

The *SCB Processing* model is MSE standard-compliant and does not require any changes to the MSE standard. However, in this model, the media data delivered to the SCB client are not ABR data. Therefore, the ABR benefits, defined in Annex B.2.2 are lost.

The *Distributed Processing* model enables the ABR on the SCB client, therefore utilizing the ABR benefits. However, this model is currently not standard-compliant and requires changes in the MSE standard.

The *Distributed Low-level Processing* model does not require any of the MSE standard changes. This model enables ABR at the SCB client and utilizes, therefore, the ABR benefits. However, this model could only be executed as a proprietary solution, which hinders technology interoperability.

For the integration of the Streaming component in the SES service, the models presented here could be chosen based on the properties that are important for a particular WMS platform. The solutions that are based on standards are favored in the industry as they provide interoperability on the market.

Therefore, as it is important to keep the ABR benefits for the end-devices, the *Distributed Processing* model might be preferred. The changes that need to be made are currently submitted for the second version of the W3C MSE standard and are under processing. These changes have been defined in this research as *solutions for the divergences* and will be analyzed further.

The solutions for the divergences discussed in the *Distributed Processing* model are evaluated in Table 6.4. All proposed solutions imply necessary changes that must be made in the current version of the MSE standard.

First, the *Divergence to be addressed* presents the divergences addressed by the solution. The solutions are evaluated on the basis of the following criteria: *Additional functionality server* shows whether the proposed solution requires additional functionality that would be executed by the WebApp running in the context of the SCB or by the SCB itself, or not; *Additional functionality client* demonstrates whether the proposed solution requires additional functionality that would be executed by the SCB client or not.

Tab. 6.4.: Evaluation of Proposed Solutions for Divergences Arising while Executing Media Source Extensions in Secure Cloud Browser

Solution	Divergence to be addressed	Additional Functionality Server	Additional Functionality Client
Execution of XHRs			
Additional XHR Type Parameters	Addresses **DIV1**: the SCB does not differentiate various XHRs. However, only the MSE-related ones must be forwarded to the SCB client.	Yes, the WebApp that runs in the SCB context must generate additional type parameters.	No
Loading of media segments			

Continued on next page

Tab. 6.4.: Evaluation of Proposed Solutions for Divergences Arising while Executing Media Source Extensions in Secure Cloud Browser (continued)

Solution	Divergence to be addressed	Additional Functionality Server	Additional Functionality Client
Additional XHR IDs	Addresses **DIV2**: as the WebApp loads data in any order, the appendBuffer method might get out of order with the SCB client.	Yes, the WebApp must generate additional IDs.	Yes, the SCB client must parse the URL in XHR to filter out the ID.
Buffering while handling the media segment			
Presentation Timestamp Ordering	Addresses **DIV3**: the SCB client does not know which media segment is being appended on the SRSE server at a certain point in time.	Yes, the WebApp must put the timestamps into XHRs.	Yes, the SCB client must filter the timestamps out and buffer the segments in exact order.
Segment Number Content Type	Addresses **DIV3**	Yes, the WebApp must include the content types in XHRs.	Yes, the SCB client must filter the content types out and play them back as audio and video data correspondingly.
Control			
Reference Data Return - Media Headers	Addresses **DIV4**: As the MSE processing is doubled, the SCB does not have any control over the data that is only available at the SCB client.	No, the WebApp uses the media headers as usual.	Yes, the SCB client must enable the return of the headers.
Reference Data Return - Parameters	Addresses **DIV4**	Yes, the WebApp must identify parameters that are required and request them at the SCB client.	Yes, the CB client must parse the media headers, filter the required parameters out and send them to the SCB.

The *Additional XHR Type Parameters* solution requires additional functionality on the server. Thus, this solution requires an additional XHR type parameter in every MSE-related request.

The *Additional XHR IDs* solution requires additional functionality on both the SCB server and client. It also requires additional logic installed on the client: the logic would control the loading of segments based on these IDs.

The adaptation of this solution requires an extension of XHRs with additional parameters, i.e. IDs. These IDs would establish the order in the media playback on the SCB client. However, this solution is directly linked to the divergences in the **Buffering while handling the media segment** stage: by addressing the divergences in this stage, the *Additional XHR IDs* solution would become redundant.

The *Presentation Timestamp Ordering* solution introduces new functionalities on both the SRSE server and client. The client also requires the buffering logic which buffers the segments according to the timestamps. This solution requires an additional parameter in XHRs. However, the *Presentation Timestamp Ordering* solution cannot be utilized if the segments are only being appended partially.

The *Segment Number Content Type* introduces new functionalities on both server and client sides. This solution extends XHRs with additional parameters. The client requires the logic of segment separation into audio and video tracks according to content type.

The *Reference Data Return - Media Headers* solution adds new functionality to the SRSE client that has to implement the logic for the return of media headers. This solution generates a significant amount of additional data in communication between the SCB and its client.

The next solution, *Reference Data Return - Parameters*, generates fewer data between the SCB and its client, as only selected parameters are sent. However, here a new functionality is required on both server and client. The client should implement the filtering logic for media headers to filter out only the requested parameter.

The degree of the solutions' complexity is high and might become a hurdle for adopting those as part of the MSE standard. However, the complexity is also partially caused by the complexity of the MSE interface itself. These solutions are however important as the main advantage of the *Distributed Processing* model is the usage of ABR benefits on the client side in a possibly standard way.

The divergences and proposed solutions defined here have been submitted by the Cloud Browser TF [W3C17b] for incorporation into the second version of the MSE standard [W3C17a].

6.2.2 Secure Streaming

The evaluation of the Security component is presented in this chapter. The approaches identified for the *Hardware-assisted EME* design are evaluated in Chapter 6.2.2.1.

Chapter 6.2.2.2 demonstrates the analysis and evaluation of the specification models of the Security component. Finally, Chapter 6.2.2.3 quantifies the impact the advanced security has on the performance of the system.

6.2.2.1. Hardware-assisted EME Approaches

The approaches of *Hardware-assisted EME*, identified in Chapter 4.2.3.1, are compared in this chapter [Sti16].

Protecting Decryption Keys in TEE: the attack model of this approach does not include protection for the content keys after they have been decrypted by the master key. Thus, it would be possible for an attacker to gain access to the content key stored in RAM in kernel space of the *Normal World* while the media being decrypted. Therefore, if an attacker is trying to get access to all content keys, such a design increases the required attack effort significantly.

With this design, the attacker has a very short window for recording the key and applying the key for content decryption before the key is erased from RAM for live streaming content. For the content that is stored on the client device, even if an attacker succeeds in recording some of these keys, he or she will get access only to some of the media data. Due to the rotation key technology, the content keys only enable access to short, i.e. 5 to 10 seconds, media segments.

This approach also keeps the code base of the TEE to a minimum, minimizing the attack surface of the TEE. This approach does not protect the Platform DRM and the Graphics Library and leaves it vulnerable in cases of attacks where adversaries compromise the OS of the *Normal World* and gain kernel privileges.

Protection of Platform DRM in TEE: the protection of the whole DRM processing within the TEE increases the overhead of TEE execution and the code base inside the TEE when compared to the first approach. As in the context of EME, multiple calls must be conducted between the Platform DRM and the WebApp, such a communication in this approach can only be executed through the constant switching between the *Secure World* and the *Normal World* that runs the WebApp.

The switching must be performed by executing the system calls defined by the TEE. This requires not only switching between the worlds but also context switches between user and kernel space. This approach leaves the Graphics Library vulnerable outside the protective environment of the TEE.

Protection of Platform DRM and Graphics Library in TEE: this approach protects all the components of the EME content decryption process. However, as the WebApp in any approach resides in the user space of the *Normal World* and has to be informed about all the EME steps happening in the platform underneath, the context and world switching also increase overhead.

In this approach the complexity of code that runs within the TEE is significantly increased. As described in Chapters 2.2.3 and 5.2.1, the Trusted OS has reduced functionality and a minimal code base. The media processing requires significant modifications to the Trusted OS.

The required protection of the system could be adjusted by the implementation of one of these approaches. The decision has to be made between the degree of protection, i.e. a number of components that are being isolated within TEE, and the code complexity within TEE that decreases the overall system protection, as only minimal code parts should be isolated in TEE.

6.2.2.2. Secure Streaming Models

Two possible specifications of the Security component in the SES service have been proposed in Chapter 4.2.3: *SCB Processing - EME* and *Hardware-assisted Distributed Processing - EME*. This chapter analyzes these models and summarizes the analysis in Table 6.5. The evaluation criteria remain identical to the ones in Chapter 6.2.1.

The *SCB Processing - EME* model is compliant with the EME standard and addresses the end-devices that do not support EME and content protection schemes of streaming source. However, with the introduction of this model, the end-to-end encryption flow is affected, as it must be interrupted at the SCB. The SCB might become a very attractive goal for adversaries. This requires additional countermeasures at the SCB.

Moreover, the unique secrets that are used in each end-device to decrypt the content must be located on one SCB server machine. This significantly increases the security requirements regarding their isolation and makes the infrastructure more complex.

Tab. 6.5.: Evaluation of Specification Models of the Security Component in Secure Streaming Service

Evaluation Criteria	Secure Cloud Browser Processing - EME	Hardware-assisted Distributed Processing - EME
Design summary	Crypto flow is completely processed by the SCB. The SCB re-encrypts the stream utilizing the encryption scheme supported by the SCB client.	EME is processed by both the SCB and the SCB client. The distributed processing is enabled by a modular EME design, where the processing modules are distributed between the SCB and the SCB client.
Requirements addressed	Requirement R8	Requirement R9
EME processing entities	WebApp, SCB	WebApp, SCB, SCB client
Awareness of EME	WebApp, SCB	WebApp, SCB, SCB client
Processing logic	Centralized: the WebApp running in context of SCB is a centralized EME processing entity that communicates with the License Server.	Distributed: the EME processing is distributed between the SCB server and the SCB client.
Divergences	None	None
Required changes in EME standard	None	None

The *Hardware-assisted Distributed Processing - EME* is also a standard compliant model. However, in real-world deployments, some of CDNs might have a single-IP requirement for security reasons[4]. As the media data is consumed by the SCB client, which has IP address 1, and the keys are requested by the WebApp, which has IP address 2, this might become a serious security violation.

The second model does not break the end-to-end encryption flow and could be successfully utilized while implementing third-party services e.g. Netflix, on the WMS platform, as these have the high-priority requirement that the distribution of content between the streaming source and the end-device cannot be interrupted. This model addresses the devices that support EME and a content protection scheme of streaming source.

Hardware-assisted EME Interface Extension The *Hardware-assisted EME* isolation analyzed in this thesis might become a part of the current EME standard[5]. EME interface is designed with the goal to give a WebApp full control over the execution flow of crypto operations. Therefore, partial or even complete code execution within TEE could also be signaled towards the WebApp as part of the standard. To enable the TEE-related events towards the browser, the following additional EME interface calls might be supported:

- *isTeeSupported* enables the WebApp to access information about whether TEE is supported on the platform.
- *TeeSupportedType* enables the WebApp to access information about which kind of TEE runs on the platform. This type of information is important for identifying the level of protection the TEE provides. This could give the WebApp information on a decision about which type of content could be served to such a platform: SD, HD, UHD, etc.

[4] According to Ronen Mizrahi, the founder and CEO of TVersity Inc. [TVe17].
[5] This proposal will be submitted by the Cloud Browser TF to be part of the EME standard version 2.

- *TeeInvoke* enables the WebApp to send the invoke command towards the platform DRM that in turn would invoke the TEE. The TEE Session ID must be provided back to the WebApp.
- *TeeInvoked event* notifies the WebApp about whether or not the TEE was successfully invoked and if it has returned the required computation results.

6.2.2.3. Measurements

Analyzing the Research Question 4, this work performs measurements of the overhead of the advanced security [Sti16]. The overhead is the impact on the system performance caused by TZ isolation of the master key.

To measure the overhead, two different measurements were taken. First, the total execution time of the TA that was called from the user space of the *Normal World* is measured.

This measurement shows the actual overhead of the TZ execution additionally introduced through the design presented in this work. Here, the time t_1 of the system clock is measured with `clock_gettime` [Opea] when the HA starts and the t_2 when HA stops the execution. As the HA executes all TA-related calls as described in Chapter 5.2, the delta t_2 - t_1 shows the total execution time of the TA.

The first group of measurements performed in the user space of the *Normal World* appeared to contain data outliers. Thus, the second group of measurements was taken in the kernel space of the *Normal World*. The ARM counter-timer kernel control register `CNTKCTL_EL1` [ARM14] in the OPTEE kernel driver is utilized for the measurement.

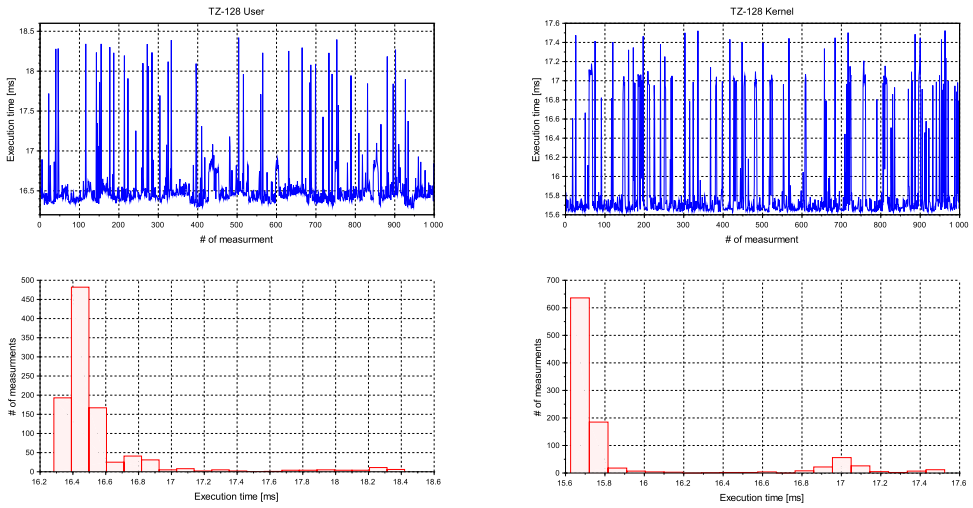
The exact number of CPU cycles required for the execution of the *SMC* is calculated by reading out this register before and after each *SMC* calls the TA. While the TA is being executed, the register is read out for all *SMCs* performed. For a total number of cycles, the CPU cycles are summed up.

To calculate the total execution time of *SMC* calls during TA execution, the total number of cycles is divided by the CPU clock frequency, defined in [96B16b]. Here, none of the *Normal World* communications are reflected in the second group of measurements. The results for two groups of measurements for a 128-bit key length AES-GCM are presented in two graphs on top of Figure 6.3a.

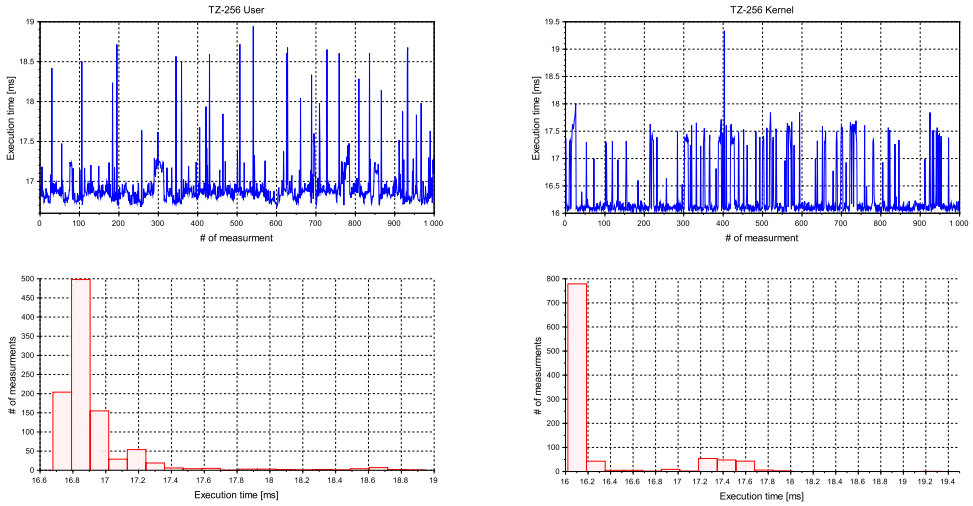
As estimated, the measurements that were taken in the kernel space of the *Normal World* have smaller values that the ones taken in the user space. This is because the communication between the HA and the TEE Supplicant is completely excluded from the second measurement presented in the top right corner of Figure 6.3a.

Nevertheless, the time characteristics are similar, and the data outliers occur in both measurements. This suggests that user space communications in the *Normal World* were not the cause of the outliers in the first group of measurements. It is assumed that these scenarios are caused by the scheduler in the Trusted OS that controls applications in the *Secure World* and the allocation of resources.

Both measurements' sets are presented through histograms on the bottom graphs of Figure 6.3a. On the histograms the best- and worst-case scenarios can be identified.



(a) Execution Time of the TA with 128-bit key length



(b) Execution Time of the TA with 256-bit key length

Fig. 6.3.: Execution Time of the TA measured from *Normal World* user space and kernel space for the (a) 128-bit and (b) 256-bit AES-GCM key lengths.

On the left graph in user space, the best-case time values are below the threshold of 17.5 ms, the values above this threshold belong to the worst case. In the kernel space, the threshold for similar scenarios is 16.4 ms.

In the user space, in the best-case scenario, the values of OCDMi execution time are increased through TZ by 16.25 ms-17.5 ms. In the worst-case scenario, they are increased by 17.5 ms-18.4 ms. In kernel space, the values are increased by 15.7 ms-16.4 ms in the best-case scenario. In the worst-case scenario, they are increased by 16.4 ms-17.52 ms.

For estimations of the actual impact on OCDMi execution time, the user space measurements are relevant. The OCDMi execution time is estimated to be 200 ms, based on the average value of timestamp measurements performed for the 128-bit key length: the start timestamps are set when the media playback session is created, the end timestamps are set when the decryption of the media content starts.

By summing up the time values with the average OCDMi execution time, the 8.12%-8.75% execution time increase can be calculated in the best case. In the worst-case scenario, the execution time increase is of 8.75%-9.2%.

The measurements for a 256-bit key length are presented in Figure 6.3b. The user space measurements are the left-sided graphs, and the kernel space are the right-sided ones. The histograms on the bottom of the figure are different forms of representation for the data shown on the top graphs.

In user space, in the best-case scenario, the values of OCDMi execution time are increased through TZ by 16.7 ms-18.0 ms. In the worst-case scenario, they are increased by 18.0 ms-18.95 ms.

In kernel space, the values are increased by 16.0 ms-16.7 ms in the best and by 16.7 ms-18.02 ms in the worst case. This is 1.83%-2.86% additional execution time, when compared to the 128-bit key length measurements in best-case, and 2.86%-2.99% in the worst case.

6.3 Secure Interactive Contents

The SIC service designed for the SRSE platform is evaluated in this chapter. Chapter 6.3.1 evaluates the models of HbbTV signaling processing and corresponding interfaces.

Based on the research results, a so-called *Integrated Approach* is presented in Chapter 6.3.2 that might address the disadvantages of the models designed in this thesis. The concepts of HbbTV execution in SRSE and corresponding interfaces that have not been addressed in the thesis will be briefly analyzed in Chapter 6.3.3.

Chapter 6.3.4 will quantify the impact of the advanced security, integrated for signaling processing, on the performance of the system. Finally, Chapter 6.3.5 will evaluate the countermeasures designed to work against the HbbTV insecurity.

6.3.1 Signaling Processing Models

The *Client-side Signaling Processing* model has been designed in specification of the functional architecture. This model addresses the client-side integration of the SIC service into SCB landscape; the *Server-side Signaling Processing* model addresses the server-side integration.

Both methods are compared in Table 6.6 according to the following criteria: the *Execution of DVB Stack* criterion analyzes where the DVB signaling processing function, i.e. DVB Stack, is executed; *Awareness of DVB* examines which system components are integrated with DVB functions; *DVB Stack required* identifies the system components that have to integrate the DVB Stack; *Regional differences of AIT* analyzes whether all types of different regional AIT tables are addressed; *Required changes in current HbbTV implementations of Red-button signaling processing* analyzes how current implementations of Red-button signaling processing are impacted; *HbbTV processing logic* examines the placement of the HbbTV signaling processing function; *HbbTV application execution* analyzes where the HbbTV applications are executed; *Devices addressed* identifies the device types that could be addressed by the model; *Trust model* describes the trust model established.

The most significant findings will be analyzed in-depth below the table.

Tab. 6.6.: Evaluation of Signaling Processing Models in Secure Interactive Contents Service: Client-side and Server-side Processing

Evaluation Criteria	Client-side Signaling Processing	Server-side Signaling Processing
Model Description	The DVB application signaling data contained in AIT is processed by the end-device. The data is then forwarded to the SCB.	The AIT is processed by the Secure Cloud Environment of the SCB server.
Execution of DVB Stack	End-device, TEE	Secure Cloud Environment
Awareness of DVB	Only end-device	SCB, Secure Cloud Environment
DVB Stack required	On the end-device	At the SCB server as part of the Secure Cloud Environment.
Regional differences of AIT	Addressed by design.	Not addressed by design.
Required changes in current HbbTV implementations of Red-button signaling processing	The functionality of the DVB Stack, currently implemented by the `oipfWebkit` library, must be divided into (1) processing and extraction and (2) filtering. The filtering must be executed within TEE.	None
HbbTV processing logic	Distributed between the end-device and the SCB.	Centralized on the SCB Server.
HbbTV application execution	SCB	SCB
Devices addressed	DVB-connected devices that implement DVB Stack, however do not support HbbTV Browser.	Any devices that do not have DVB capabilities and do not support HbbTV.
Trust model	AIT filtering function is only trusted if executed within TEE as the end-device is located on user premises.	AIT filtering function is trusted as executed by the SCB server on content-provider premises.

In the *Client-side Signaling Processing* model, the regional differences of the AIT table are addressed by design, as the DVB signal available in a certain region will be processed at the end-device located in this region. The regional differences in broadcasted signal might come in the broadcasting of local advertisements or local news. The server-sided model does not provide this advantage, and all different types of AITs should be processed by the SCHB server, which requires more execution logic.

In the *Client-side Signaling Processing* model, the availability of the DVB data at the end-device poses security risks to the system. In this case, the integrity of the signal must be ensured. Assuming that an attacker can compromise the end-device and insert malicious data into the AIT, the AIT processing must be performed in TEE.

Integration of TEE environments requires a large engineering overhead and is very platform dependent. In the *Server-side Signaling Processing* model, the TEE integration is not required as the SCHB platform is considered to be trusted.

In the *Server-side Signaling Processing* model, the required processing of the DVB Stack within the Secure Cloud Environment might have insufficient support, which could make such a model infeasible. This has been seen in the operator domain where operators struggle to integrate the Multicast stack into off-the-shelf OpenStack[6] technology with no success thus far [V. 16].

Cloud computing environments are built on general purpose hardware and software platforms; custom adjustments increase the costs of Cloud infrastructures, turning them into

[6]OpenStack [Opeb] is an open source Cloud operating system that enables provisioning of resources, which is very cost-effective off-the-shelf.

non-cost-effective systems. Similarly, currently, there is no support for the DVB Stack in off-the-shelf Cloud orchestrating technologies, it might be the main obstacle on the way to the integration of the *Server-side Signaling Processing* model into the Cloud.

In the *Client-side Signaling Processing* model, the question of which role the streaming provider takes in such scenarios also remains open. The regulations and industry will decide whether or not the provider is allowed to intercept the HbbTV Red-Button signaling.

6.3.1.1. Signaling Processing Interfaces

The *Client-side Signaling Processing* model might be processed over the interfaces between the SCHB and its client. This would be a part of the functionality of the *Cloud Browser API*, which is currently undergoing processing in the W3C Cloud Browser TF. Within the specification of the SIC service, this interface has been identified as the `Server:Client` interface.

The `Server:Client` interface does not require HbbTV adaptations and enables a full integration of SoA HbbTV into SCB. The HbbTV Application executed within the SCHB context will not have an awareness of being executed remotely.

Such methods as *getChannelID* and *getApplicationUrl* might be supported by the `Server:Client` interface. The *getChannelID* method would enable the SCHB to query its client for the channel that is currently tuned by the tuner. The *getApplicationUrl* method enables the SCHB to request the *HbbTV Application URL* at the SCHB client according to the previously received channel ID.

The *Client-side Signaling Processing* model might also be partly processed over currently existing HbbTV interfaces defined by OITF in [OIP14] and now adapted in [ETS16] by HbbTV, due to the deprecation of the OITF. However, for this adaptations of the currently existing HbbTV interfaces are needed.

For example, the `createApplication` method in the Application Management API implemented by the HbbTV Browser might be extended. Currently this method only uses the parameter `url_of_application` that takes the *HbbTV Application URL* as a value. It might be extended with parameters specifying the SCHB client properties, e.g. client ID, as the URLs are physically available at the end-device and not on the server. Having the required client ID, the SCHB server would be able to request the *HbbTV Application URL* and make it available to the SCHB.

6.3.2 Integrated Approach

Considering that both models have pros and cons, an integrated approach might be the best choice for integrating HbbTV into the SCHB landscape. HbbTV Browser is usually integrated with HbbTV-specific libraries, i.e. DVB Stack comprising the `oipfWebkit` and `oipfDSMCC` libraries, for extracting and requesting the signaling data. Regarding the current implementations of HbbTV as stated in Table 6.6, in order to process the AIT data securely, the currently available DVB stacks must be divided in their functionality.

The (2) filtering, identified in Table 6.6, has to be placed within the TEE to enable the integrity of data that is sent to Cloud environments. Regarding the complexity of TEE

integration and the fact that the DVB functionality must be divided either way, the best approach, i.e. *Integrated Approach*, might be the placement of the filtering function onto the SCHB server. Though this would require a *doubling* of the signaling processing functionality, it also has its advantages: the AIT table received by the end-device could be easily forwarded to the SCHB.

In this case, the verification of the AIT signature and the filtering of the *HbbTV Application URLs* would be processed within the SCHB server. Thus, the verification, filtering and signing functions will not be required on the end-device. Consequently, none of the isolation technologies, e.g. TEE, as utilized in this thesis, would be required on the end-device. This would simplify the end-device functionality.

However, this would also require the placement of the filtering functionality, currently implemented by the the `oipfWebkit` library, on the SCHB server. The libraries are part of the DVB stack and therefore the integration of the DVB stack into the Secure Cloud Environment would be required.

According to the previous analysis, the integration of custom libraries into the Secure Cloud Environment might cause difficulties. To solve this problem, only the filtering function might be integrated into the HbbTV Runtime Environment. This could be done, as the filtering function is not closely integrated with the actual processing of the broadcast stream and only performs operations with the AIT data, which could be executed by high-level Web languages.

Such integration into the HbbTV Runtime Environment might be done either in a proprietary way or be a part of a standard functionality, which would require adaptations in HbbTV standard. Regarding interface adaptation, none of the adaptations of the currently existing HbbTV interfaces would be required: the AIT table can be simply pushed towards the SCHB as part of standard server-client communication.

6.3.3 HbbTV Execution

This chapter addresses the *Interactive Data Functions* and *Interactive Application Functions* groups identified in Chapter 4.3.1. These functions are not within the scope of this thesis and have therefore not been addressed so far; however, they are relevant for the execution of HbbTV applications.

HbbTV execution is closely interconnected with the signaling processing studied in this thesis and the results of the research would have a direct impact on HbbTV execution, therefore it will be briefly considered in this chapter.

Interactive Data Functions consist of **requesting** and **delivery** functions that can be executed over broadcast or broadband interfaces, dependent on the URL prefix. Requesting is done over the broadcast interface if the prefix is `dvb://` and will be referred as **broadcast requesting**.

These requests are executed by the `oipfDSMCC` library that implements the DSM-CC URL Handler. The DSM-CC URL Handler waits for the requested data to appear in the DSM-CC carousel and fetches the data that is then delivered to the end-device over the broadcast interface accordingly. This function will be referred to as **broadcast delivery**.

In the case of the prefix `http(s)://` the request takes place over the broadband interface. The request is executed by the TCP/IP stack of the platform. The data is delivered to the end-device over the broadband interface. Both functions will be referred to as **broadband requesting** and **broadband delivery**.

After reception the application data is executed by the functions in the *Interactive Application Functions* group. To perform execution and composition functions in the case of HbbTV a so-called HbbTV-enabled Browser, i.e. HbbTV Browser, and Application Manager are used.

The browser implements the communication towards the libraries described previously e.g. for extracting and requesting the AIT data. It also adopts the HbbTV-specific integrations and interfaces of the specifications listed in Annex B.4.1.

The Application Manager performs an AIT evaluation to control the life-cycle of HbbTV applications. The Application Manager is part of the HbbTV profile of the Browser and together with the HbbTV Browser comprises the HbbTV Runtime Environment.

In the *Server-side Signaling Processing* model, the DVB Stack availability is given on the server. Therefore both function groups *Interactive Data Functions* and *Interactive Application Functions* are placed on the server. The requesting and delivery functions are executed over both broadcast and broadband interfaces. This model would therefore fit into the Single Stream SCB approach.

In the *Client-side Signaling Processing* model, the DVB Stack resides on the end-device, and the processing of the broadcast signal is executed on the end-device. Thus, the signal that also delivers media content and application data is available on the end-device at all times, which would make the re-usage of media-related client functions, e.g. media rendering, rational. This model could therefore be applied to the Double Stream SCB approach.

In this model, the broadcast requesting and broadcast delivery functions are executed over the broadcast interface by the end-device, as this interface is available at the end-device. The media data is executed by the media-related functions on the end-device. The application data is forwarded to the SCHB.

In the case of sensitive application data, an integration with the TEE, similar to the integration for isolating signaling data performed in this thesis, could be established. The broadband requesting and broadband delivery functions are then placed on the server, as the broadband interface is also available at the server. This function placement is also valid for the Integrated Approach without the DVB Stack on the server defined in Chapter 6.3.2.

This execution in the context of the Double Stream approach enables the reuse of legacy media delivery networks. In the case of interactive services, this would enable the reuse of broadcaster networks for media delivery, reducing the overall SRSE network load while broadcasting live events to a high amount of end-users.

Regarding the HbbTV execution, the Integrated Approach, addressing the Double Stream delivery, might also be the best choice for HbbTV integration into SCHB landscape.

6.3.3.1. HbbTV Execution Interfaces

The OIPF DAE specification [OIP14] defines the JS APIs needed for TV applications to access platform functionality. Currently this specification has been adapted in [ETS16] and in an Operator Applications specification announced in [Hbb] by HbbTV.

These APIs address the entire functionality of the IPTV services and the most important APIs include: Application Management API, Content Service Protection API, Media Playback API and Scheduled content and hybrid tuner APIs. To enable HbbTV for SCHB in a way that could be adopted by the HbbTV Standard, the APIs must be adjusted. The Application Management API is responsible for controlling and management of applications and has been addressed in Chapter 6.3.1.1.

To replace the Content Service Protection API, the HbbTV in [ETS16] references the W3C EME interface for the playback of the encrypted content. This interface, as analyzed throughout the research for this work, can be easily executed in the SCHB landscape for both Single and Double Stream approaches.

It is possible that also the Media Playback API in the near future will be replaced by the W3C MSE interface. The MSE interface comprises complex techniques and requires multiple adaptations if it is to be integrated into the Double Stream SCHB. The Scheduled content and hybrid tuner APIs also would need certain adaptations as the media is only physically available on the end-device. However, this interface has not been considered in this thesis and is beyond the scope of this research.

In summary, the existing HbbTV JS interfaces do not require adaptation regarding the Single Stream SCHB approach. In this approach, the whole execution is done on the server side the same way it is currently done on end-devices. They, however, require adaptations when considering the Double Stream approach as the functionalities are distributed between the server and the client. In this approach, the physical data is no longer available on the server side, which would require adaptations in existing interfaces.

6.3.4 Measurements

The advanced security introduced in the design of the SIC service is addressed in this thesis through the integration of certain end-devices functionality into the TZ. The execution time of such advanced security is analyzed in this chapter to answer the Research Question 5 posed in Chapter 1.4.

The device functionality isolated in the TZ included verification, filtering and signing of the AIT data. The time needed for the execution of this functionality within the TZ is measured in this section. To measure the execution time, three different set of measurements are executed. The results of the measurements are presented in Figure 6.4.

First, the total execution time of the TA called from the user space of the *Normal World* is measured. In this measurement, none of the functions listed above are executed by the TA. This measurement shows the round trip time needed to switch from the *Normal World* into the *Secure World* and get back into the *Normal World*. Thus, the overhead of the TZ execution integrated additionally into this work is shown.

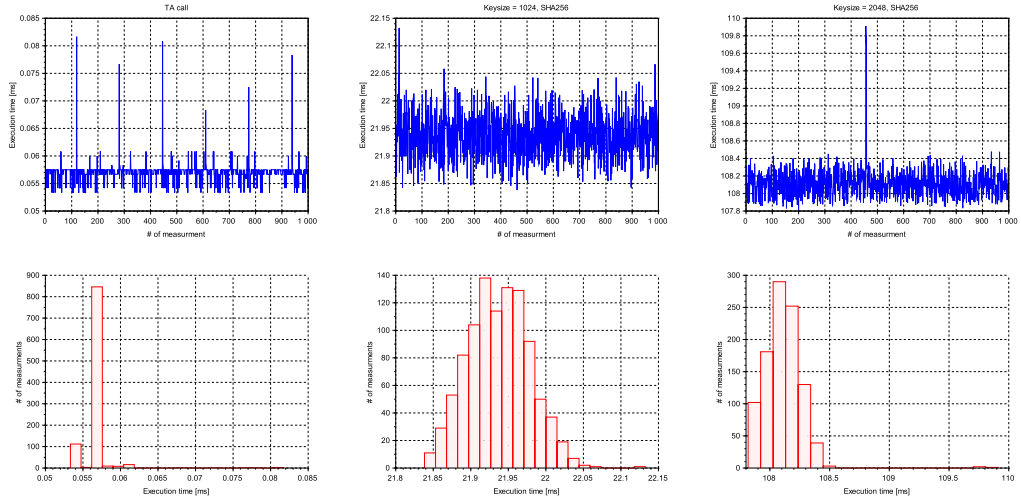


Fig. 6.4.: Execution Time of the Trusted Application measured from *Normal World* user space for the (left) TA call, (center) SHA256 RSA-1024 and (right) SHA256 RSA-2048 in the *Hardware-assisted Red-button Signaling Processing*

The time t_1 of the system clock is measured with `clock_gettime` when the HA starts within the SCHB client. The time t_2 is measured when the HA stops its execution. The execution time of the HA t_2 - t_1 measures the exact overhead of the TZ execution as the HA executes all TA-related calls, as described in Chapter 5.3. The results of the first measurement are presented at the top of Figure 6.4 (left). The measurements are performed 1000 times, as shown on the x-axis.

The same data set is represented by the histogram on the bottom of Figure 6.4 (left), where it can be seen that the most probable execution time is 0.057 ms. The data outliers are rather rare and are assumed to be caused by *Normal World* communications.

In the second and third measurements, all three functions - verification, filtering, and signing - are executed. The measurement's set up is identical to the first measurement: time stamps t_1 and t_2 are set in the same manner.

In the practical implementation, the SHA256 algorithm was used for hashing in both verification and signing functions. In both functions, the same encryption algorithm was applied. The measurements for RSA-1024 bit key length and RSA-2048 bit key length are presented in Figure 6.4 (center) and (right) correspondingly.

The corresponding histogram representation shows that in value distribution the most probable values are 21.92 ms for RSA-1024 and 108.1 ms for RSA-2048.

The measurements show that the actual context switch between the two worlds in the TZ adds almost no overhead to the system execution. The execution time of the functions within TA using the 2048 key length is 4,93 times more than using the 1024 key length.

The overall system latency time was not measured in the measurement set-up, as the actual HbbTV execution was out of scope in this implementation. The system latency time identified by Doherty and Thadhani in [Doh82] is used for comparison.

In their work, they identify 400 ms as the threshold value for overall system latency. The values under this threshold[7] for the overall system latency make the system attractive for the end-user.

According to this value, the execution times are compared to estimate the relations of the results to those required by the market system latency. Compared to this acceptable overall latency value, the time execution of the signaling on the end-device is 5,48% of the overall system latency for a 1024 bit key.

It comprises, however, 27% for 2048 bit key, which is a considerable amount. Thus, based on these results, the 1024 key length might be considered as a first choice, if it complies with security requirements of the system.

6.3.5 Authentication of a Broadcaster

At the time of writing the thesis, none of the technologies for authentication of the broadcaster streams against the end-devices were available in HbbTV. The vulnerabilities in HbbTV caused by the missing authentication are introduced in detail in Chapter 2.3.4. It is also specified there that the countermeasures proposed to mitigate this issue in [Mic15] are set to be adapted and standardized by DVB in 2017.

The research in this thesis assumes that the broadcaster streams are signed by broadcasters. The technology that is planned to be used for this is identified in [Hbb15] and adapted into this research without changes. This ensures a compatibility of presented solutions with future versions of HbbTV that will include such countermeasures. To preserve compatibility, in the designed solution the filtered AIT data is signed within the TEE. However, the data could also be encrypted within the TEE, depending on platform requirements.

[7] This value has been used by the author of the thesis in projects from Deutsche Telekom as a threshold value for the maximum possible overall latency of the SRSE platform [A. 16].

7 Conclusion

A novel architecture for *Secure Remote Service Execution (SRSE)* has been presented in this thesis. In this architecture, the Media Service Execution Environment (MSEE) is decoupled from the hardware of the local platform through a secure remote execution that resolves hardware dependencies. This makes the system highly adaptive and flexible, and allows it to provide a variety of application environments.

The architecture solves the problem of the limitations of end-devices' capabilities and thus enables content providers to deliver new, rich, multimedia services to low-end devices.

Using this design, a uniform Media User Interface (MUI) can be provided to a broad range of devices, by simply creating a custom Web application that is processed by the SRSE and delivered as a video stream to the end-device. Furthermore, it reduces the need for processing power inside the end-device and helps deploy new MUI technologies faster.

This work presented a detailed analysis of the current State-of-the-Art (SoA) technologies and related works. Over the course of the analysis, currently established MSEEs, *streaming*, *protected streaming* and *interactive service* technologies were analyzed. Throughout this analysis, the following technologies were selected for the specification of the SRSE architecture:

- the Web browser was identified as an MSEE for the specification of the SRSE;
- the *streaming* and *protected streaming* technologies that have been selected for integration with SRSE are W3C Media Source Extensions (MSE) [Col+14] and Encrypted Media Extensions (EME) [Dor+13] interfaces respectively;
- the *interactive service* technology selected is the European HbbTV standard [ETS16] for interactive TV.

To design the architecture, first, an analysis of requirements was performed. Based on the identified requirements, the functional architecture SRSE was designed to perform the shift of the local MSEE into the Cloud.

In this step the functional architectures for the *streaming*, *protected streaming* and *interactive service* were also enabled for remote execution in the context of SRSE. These architectures are referred to as *Secure Streaming (SES)* and *Secure Interactive Contents (SIC)* services, respectively.

Afterward, the detailed specifications for the three identified architectures were derived: the specification for *Secure Cloud Browser (SCB)* as a special case of the SRSE, the *Secure Streaming* service enabling execution of MSE and EME integrated with TEE and *Secure Interactive Contents* service enabling execution of HbbTV integrated with TEE.

The focus of the practical implementation has been defined throughout the design of the specifications. The following two systems were implemented: (1) *Hardware-assisted EME* and (2) *Hardware-assisted Red-button Signaling*. These implementations integrate EME and

HbbTV Red-button signaling respectively with the Trusted Execution Environment (TEE) implementation of ARM, i.e. ARM TrustZone [ARM17a], to isolate sensitive platform data within the TEE.

Countermeasures have been applied to protect the system against the security vulnerabilities of EME and HbbTV identified earlier in the analysis of SoA technologies and related works. Finally, the design of the architecture and practical implementations have been evaluated through a detailed comparative analysis of the various architectural models considered throughout the research and the corresponding measurements.

7.1 Summary of Research

Research Question 1: How can the shift of local MSEEs into the Cloud be executed with a design for the *SRSE* that preserves the security of high-value content?

The modeling of the shift of the local MSEE to the Cloud performed in this thesis was conducted at first completely abstracted from any MSEE technology that currently exists. The MSEE was broken down to the most important functions that are part of every current MSEE. These functions were then placed onto the remote server or preserved on the end-device, depending on the approach considered.

With regard to security functions that enable content protection of high-value media, this thesis presented two approaches. The *Single Stream SRSE* approach proposes the processing of the content protection scheme of streaming source entirely on the SRSE server. This is followed by re-encryption of the content to be securely delivered to the end-device.

The *Double Stream SRSE* approach enables end-to-end content protection between the key servers and the end-device, where the SRSE server has a controlling role in the actions performed while decrypting the content locally on the client.

The content protection function cannot be shifted into the Cloud, as this is the crypto function and requires local execution on the end-device in all of the approaches. Thus, the entire functionality cannot be shifted as the content protection function must always reside on the client.

Research Question 2: What impact does the SRSE model have on current network infrastructures?

The shift of the maximum MSEE functionality into the Cloud presented in this thesis is reflected in the *Single Stream SRSE* approach. This approach enables maximum flexibility as the service execution logic resides completely on the SRSE server and the end-device is only responsible for security functionality.

However, the network load model built for this approach displayed the fact that it's not applicable for existent network infrastructures as it requires a huge amount of bandwidth. It also requires a big capital investment to locate the SRSE service close to the consumer for better user experience, as such a placement would require a huge amount of SRSE nodes.

This issue can be resolved through the second approach *Double Stream SRSE*, where only the MUI is handled by the SRSE server and the complete media processing is located on the end-device. With the *Double Stream SRSE* approach, the legacy delivery mechanisms, e.g. IP

multicast, cable, etc., can be re-used for Live TV broadcast delivery thus resulting in less additional load on the network.

Research Question 3: How can *streaming* be realized for the SRSE platform? Which new interfaces must be defined for this realization?

The MSEE selected in this research for a detailed specification of the SRSE is the Web browser. Herewith, the *streaming* technology addressed is the W3C MSE interface that enables the Web browser to consume Adaptive Bitrate streaming.

Three different models have been proposed in this thesis to enable MSE in the context of the SCB. The *SCB Processing* model enables the processing of the entire MSE logic on the SCB server. The *Distributed Processing* model addresses MSE processing through the *doubling* of the processing, where both processing entities, the SCB and the SCB client, execute the MSE with the *only* difference being that the SCB client does not have any awareness of MSE.

The *Distributed Low-level Processing* model proposes the introduction of the low-level interface between the SCB server and the SCB client that would enable the execution of the MSE on both entities by stretching the MSE interface over the network.

In this thesis, the *Distributed Processing* model is assumed to be the most promising approach. The logic behind (1) *the interface that enables MSE communication between the SCB and its client* and (2) *the MSE interface* differs from approach to approach and has been defined in this research. The approaches developed in this research for (1) and (2) interfaces have been submitted to the W3C as *Cloud Browser API* and *MSE Version 2* respectively.

Research Question 4: How can *protected streaming* be realized for the SRSE platform? Which new interfaces must be defined for this realization? Is it possible to enhance the security of *protected streaming* through the use of a TEE that isolates sensitive data? What would the overhead of such advanced security be?

The *protected streaming* technology selected in this thesis is the W3C EME interface. EME enables the Web browser to consume encrypted media content. The two models that address the problem of remote EME execution in the context of the SRSE were presented here. The *SCB Processing - EME* model proposes a complete EME processing on the SRSE server, followed by the re-encryption of the content for secure delivery to the end-device.

The *Hardware-assisted Distributed Processing - EME* model stretches the EME interface between the SCB server and the SCB client in a secured manner. These models are enabled over the interface defined in this work, which enables EME communication between the SCB and its client as part of the *Cloud Browser API*.

In addition, this thesis developed the *Hardware-assisted EME* integration of EME with a TEE on the end-device, in order to isolate the content decryption key within the TEE. The quantitative analysis of the overhead that this advanced security adds to the overall system performance has been carried out and the increase in the execution time has been identified to be under 10% when compared to the EME processing without TEE.

Research Question 5: How can *interactive service* be enabled for SRSE? Which interfaces should be defined for remote execution? Is it possible to protect such a system through TEE against the attacks defined earlier? What is the additional execution time for such security measures?

The *interactive service* technology selected in this thesis is the HbbTV standard. The remote execution model for HbbTV developed in this thesis concentrated on the processing of the HbbTV signaling data on the end-device, the so-called *Client-side Signaling Processing* model. After processing on the end-device, the data is sent to the Secure Cloud HbbTV Browser (SCHB) server. The SCHB executed on the server is the HbbTV Browser that integrates the HbbTV libraries required for HbbTV application execution.

The *Server-side Signaling Processing* model, where the HbbTV data is processed by the server, has not been presented in the research in detail. However, both approaches have been analyzed comparatively throughout the evaluation.

The first model is considered to be the most viable for devices with DVB support. This model also addresses the regional differences in the HbbTV signaling data by design and does not add more unnecessary complexity to the SCHB server.

However, the second model addresses devices that do not even support HbbTV, which would be a perfect solution for HbbTV in mobile domain. The newly defined interface proposed in this thesis is based on the local processing of the HbbTV data, resides between the SCHB server and the SCHB client, and is part of the *Cloud Browser API*.

This thesis also analyzes the threats involved in such client-side processing of the HbbTV Red-button signaling data. As it is possible to tamper with the *HbbTV Application URLs*, the logic of URL processing has been shifted to the TEE and extended with additional verification and signing algorithms in the so-called *Hardware-assisted Red-button Signaling* model.

The execution time measured was 21.92 ms for RSA-1024 and 108.1 ms for RSA-2048, which is 5,48% and 27% respectively, when compared to the overall accepted system latency that should be under 400 ms.

7.2 Future Research

This section presents future research directions and possible applications for the SRSE architectures developed in this thesis, in particular the SCB.

Dynamic Secure Cloud Browser

The detailed analysis of SCB approaches has shown that each of these approaches addresses certain shortcomings with current Web streaming platforms. While the Single Stream approach enables the creation of the whole MUI together with the media in the Cloud, the Double Stream approach saves on network bandwidth and computing resources, and delivers the media to the end-device separately.

Both approaches bring the latest Web technologies to legacy end-device platforms and could be deployed jointly in one system. One of the possible research directions in the deployment of these approaches in a more flexible way is a *flexible function distribution* for SCB.

In such an architecture, the functions are not initially defined with any parameters or properties[1]. Throughout the process of initialization, the localization of functions is assigned by defining whether or not they are currently placed on the server- or client-side.

The initialization process could be repeated, and the localization of functions and correspondingly their parameters and properties could be dynamically reassigned. This would enable the redistribution of functions on-the-fly between the SCB and its client.

The so-called *Dynamic SCB* would enable intelligent resource and bandwidth consumption, better reaction to resource shortage, fast adaptation to multi-device environments and better targeting of Web applications that might have different requirements.

In the context of the *Dynamic SCB*, even within a single function, such a flexible distribution might be developed. This could be used for the combination of the *Cloud MUI* with the *Local MUI* for a more efficient MUI execution.

Single Stream as Single Solution

The proposed *Single Stream SRSE* approach requires much more bandwidth and computing resources when compared to the *Double Stream SRSE* one. Therefore, with regard to bandwidth consumption, the latter approach might be currently of interest when deploying Web streaming platforms.

However, considering the pace of development in current Cloud storage, Cloud computing and available bandwidth technologies, the *Single Stream* approach might become the preferred one over the long-term.

In addition, new highly efficient technologies like H.265/HEVC for stitching together multiple streams, which do not require transcoding, might become SoA technologies in the near future. Such architectures, as analyzed in Paper 9, decrease the required computational power on the server-side that might in turn promote the *Single Stream* deployments in the future.

Shift of Applications into the Cloud

With the advent of powerful Cloud computations and available Cloud platforms, the shift of applications into the Cloud is constantly gaining momentum. Cloud costs and the overhead of the Cloud computing are continuing to decrease and overtaking the increasing complications involved in native application development. This results in different models of Web-hosted applications, where the content resides in the Cloud.

The projects such as Apache Cordova [Apa17] or Microsoft Xamarin [Mic17h] try to decrease application development overhead by enabling application execution on multiple platforms after developing it once. Besides, to enable execution of Web sites on multiple platforms, W3C is currently working on a Web App Manifests [W3Ca] standard to allow websites to declare their application-like properties. Frameworks like Manifold.JS [Man17] use this standard for platforms where the support of Web App Manifest is provided.

In the context of the application shift into the Cloud and with the constantly increasing fragmentation of consumer devices, the SCB might become one of the future directions for application deployment. The SCB might also become part of the projects mentioned above,

[1] According to Colin Meerveld, the platform architect at ActiveVideo [Act].

in which not only the content itself but also the execution of the applications might be relocated into the Cloud to address any end-user devices.

Web Development and Future of HTML

Web technologies, evolving at high speed, demand more frequent upgrades of end-user device capabilities. Due to the significant investments that these upgrades require, the gap between device hardware and Web requirements is increasing. The SCB might bridge this gap in the future by abstracting the user experience of hardware device capabilities.

Another important project, WAVE, is currently providing a standard Runtime Environment (RTE) for media playback that would be supported by the major platforms. Based on WAVE, the W3C Web Media APIs group [W3Cb] has been established, with the aim of standardizing such an RTE and providing a reference player implementation. The SCB client might become part of this specification, enabling the SCB architecture by default on any streaming platform.

Further Execution Environments

This thesis focused on the Web browser to specify the SRSE. Analog to the SCB specification developed, other MSEEs might be implemented remotely. Such an MSEE might also become an Android OS and provide the *Secure Cloud Android OS*, where, analogue to the SCB, the Android OS would be executed on a remote machine. This would solve the problem of Android OS versioning that currently presents a big challenge in addition to the highly fragmented android device market.

Remote Service Execution Mobile

Considering the rapid developments of mobile platforms and their high fragmentation, the SRSE might find its largest rate of deployment in the mobile domain. Web browser and Android OS versioning might be addressed by the SRSE to speed up innovation, ease the deployment process, decrease the number of security patches, etc. The open issue that will have to be solved here is that of efficient bandwidth usage, as in the mobile world that might still be an issue in the near future. This issue could be addressed by currently evolving 5th generation mobile networks, i.e. 5G networks, that enable nearly unlimited bandwidth and low-latency computing enabled through distributed edge-computing.

Bibliography

[28996] Technical Report 289. *Digital Video Broadcasting (DVB); Support for Use of Scrambling and Conditional Access (CA) within Digital Broadcasting Systems*. 1996.

[Aci+09] O. Aciicmez, J. P. Seifert, and X. Zhang. „A Secure DVB Set-Top Box via Trusting Computing Technologies“. In: *2009 6th IEEE Consumer Communications and Networking Conference*. 2009, pp. 1–8 (cit. on pp. 41, 173–175).

[AJ+83] S. R. Ames Jr, M. Gasser, and R. R. Schell. „Security Kernel Design and Implementation: an Introduction.“ In: *IEEE computer* 16.7 (1983), pp. 14–22 (cit. on p. 29).

[Ana+13] I. Anati, S. Gueron, S. Johnson, and V. Scarlata. „Innovative Technology for CPU Based Attestation and Sealing“. In: *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy* 13 (2013), pp. 1–7.

[Ath+15] M. Athanasakis, E. Athanasopoulos, and M. Polychronakis. „The Devil is in the Constants: Bypassing Defenses in Browser JIT Engines.“ In: *NDSS* (2015) (cit. on pp. 2, 17).

[Ban+10] S. Bandhakavi, S. T. King, and P. Madhusudan. „VEX: Vetting Browser Extensions for Security Vulnerabilities.“ In: *USENIX Security Symposium*. Vol. 10. 2010, pp. 339–354 (cit. on p. 6).

[Bar+10] A. Barth, A. Porter Felt, P. Saxena, and A. Boodman. „Protecting Browsers from Extension Vulnerabilities“. In: *Proceedings of the Network and Distributed System Security Symposium, NDSS 2010, San Diego, California, USA, 28th February - 3rd March 2010*. 2010 (cit. on p. 6).

[Ben14] G. Bengtz. „Analysis of New and Alternative Encryption Algorithms and Scrambling Methods for Digital TV and Implementation of a New Scrambling Algorithm (AES128) on FPGA“. MA thesis. Sweden: Linköping University. Department of Electrical Engineering., 2014 (cit. on p. 159).

[Ben16a] G. Beniamini. *Extracting Qualcomm's KeyMaster Keys - Breaking Android Full Disk Encryption*. July 2016 (cit. on p. 42).

[Ben16b] G. Beniamini. *QSEE Privilege Escalation Vulnerability and Exploit (CVE-2015-6639)*. May 2016 (cit. on p. 42).

[BH06] J. Bishop and N. Horspool. „Cross-Platform Development: Software that Lasts“. In: *Computer* 39.10 (2006), pp. 26–35 (cit. on p. 1).

[Cal+13] G. M. Calixto, A. C. B. Angeluci, L. C. P. Costa, R. de Deus Lopes, and M. K. Zuffo. „Cloud Computing Applied to the Development of Global Hybrid Services and Applications for Interactive TV“. In: *2013 IEEE International Symposium on Consumer Electronics (ISCE)*. 2013, pp. 283–284.

[Cao+12] B. Cao, J. Yin, S. Deng, Y. Xu, Y. Xiao, and Z. Wu. „A Highly Efficient Cloud-based Architecture for Large-scale STB Event Processing: Industry Article“. In: *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*. DEBS '12. Berlin, Germany: ACM, 2012, pp. 314–323.

[CC09] M. Chase and S. S. M. Chow. „Improving Privacy and Security in Multi-authority Attribute-based Encryption“. In: *Proceedings of the 16th ACM Conference on Computer and Communications Security*. CCS '09. Chicago, Illinois, USA: ACM, 2009, pp. 121–130 (cit. on p. 26).

[Che+10] Y. Chen, V. Paxson, and R. H. Katz. „What's New About Cloud Computing Security?“ In: UCB/EECS-2010-5. 2010 (cit. on pp. 39, 171, 172).

[Che14] B. Cheng. „MediaPaaS: A Cloud-Based Media Processing Platform for Elastic Live Broadcasting“. In: *2014 IEEE 7th International Conference on Cloud Computing (CLOUD)*. 2014, pp. 713–720 (cit. on pp. 43, 176, 177).

[Cho+09] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, and J. Molina. „Controlling Data in the Cloud: Outsourcing Computation Without Outsourcing Control“. In: *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*. CCSW '09. Chicago, Illinois, USA: ACM, 2009, pp. 85–90 (cit. on pp. 38, 171, 172).

[Chu+11] B. G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti. „CloneCloud: Elastic Execution Between Mobile Device and Cloud“. In: *Proceedings of the Sixth Conference on Computer Systems*. EuroSys '11. Salzburg, Austria: ACM, 2011, pp. 301–314 (cit. on pp. 38, 171, 172).

[CM09] B.-G. Chun and P. Maniatis. „Augmented Smartphone Applications Through Clone Cloud Execution“. In: *Proceedings of the 12th Conference on Hot Topics in Operating Systems*. HotOS'09. Switzerland: USENIX Association, 2009, pp. 8–8 (cit. on p. 38).

[Col+14] A. Colwell, A. Bateman, and M. Watson. *Colwell: Media Source Extensions*. W3C Candidate Recommendation, 2014 (cit. on pp. 5, 21, 123).

[Con98] US Congress. „Digital Millennium Copyright Act“. In: *Public Law* 105.304 (1998), p. 112 (cit. on p. 26).

[Cos+15] V. Costan, I. Lebedev, and S. Devadas. „Sanctum: Minimal RISC Extensions for Isolated Execution“. In: MIT CSAIL (2015) (cit. on p. 41).

[Css] *CSS Snapshot*. W3C, Jan. 2017 (cit. on p. 14).

[Cut97] D. J. Cutts. „DVB Conditional Access“. In: *Electronics and Communication Engineering Journal* (1997) (cit. on p. 157).

[Def+14] D. Defreez, B. Shastry, H. Chen, and J. P. Seifert. „A First Look at Firefox OS Security“. In: *CoRR* abs/1410.7754 (2014).

[Dej+14] D. Dejanović, M. Subotić, N. Fimić, and L. Benarik. „A Method of Centralized Resource Management on a Set-Top Box“. In: *2014 IEEE Fourth International Conference on Consumer Electronics Berlin (ICCE-Berlin)*. 2014, pp. 144–146 (cit. on p. 61).

[Die12] E. Diehl. *Securing Digital Video: Techniques for DRM and Content Protection*. Springer Science & Business Media, 2012 (cit. on p. 157).

[DN14] C. Dall and J. Nieh. „KVM/ARM: The Design and Implementation of the Linux ARM Hypervisor“. In: *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '14. Salt Lake City, Utah, USA: ACM, 2014, pp. 333–348 (cit. on p. 30).

[Dom] *Document Object Model (DOM)*. W3C, Jan. 2005 (cit. on p. 16).

[Dor+13] D. Dorwin, A. Bateman, and M. Watson. *Encrypted Media Extensions*. W3C Candidate Recommendation, 2013 (cit. on pp. 5, 123).

[Dor+14] D. Dorwin, A. Bateman, M. Watson, and J. Smith. *ISO Common Encryption EME Stream Format and Initialization Data*. W3C, Aug. 2014 (cit. on p. 162).

[DS+11] D. Diaz-Sanchez, F. Almenares, A. Marin, D. Proserpio, and I. Telemática. „Media Cloud: Sharing Contents in the Large“. In: *2011 IEEE International Conference on Consumer Electronics (ICCE)*. 2011, pp. 227–228 (cit. on p. 38).

[Duf+11] J.-C. Dufourd, S. Thomas, and C. Concolato. „Recording and Delivery of HbbTV Applications“. In: *Proceedings of the 9th International Interactive Conference on Interactive Television*. EuroITV '11. Lisbon, Portugal: ACM, 2011, pp. 51–54.

[Ege+09] M. Egele, P. Wurzinger, C. Kruegel, and E. Kirda. „Defending Browsers against Drive-by Downloads: Mitigating Heap-Spraying Code Injection Attacks“. In: *Detection of Intrusions and Malware, and Vulnerability Assessment*. Berlin, Heidelberg: Springer Berlin Heidelberg, July 2009, pp. 88–106 (cit. on p. 2).

[Ekb13] J.-E. Ekberg. *Securing Software Architectures for Trusted Processor Environments*. Aalto University, 2013 (cit. on pp. 28, 30).

[Fra+14] J. Franke, A. Beyer, and B. Cheng. „Virtual Set-Top Box in the Cloud: Enabling Interactive Third Party Applications“. In: *2014 IEEE Network Operations and Management Symposium (NOMS)*. 2014, pp. 1–4 (cit. on pp. 43, 176, 177).

[Fre+10] T. Frenzel, A. Lackorzynski, A. Warg, and H. Härtig. „ARM TrustZone as a Virtualization Technique in Embedded Systems“. In: *Proceedings of Twelfth Real-Time Linux Workshop, Nairobi, Kenya*. 2010 (cit. on pp. 29, 42, 173–175).

[Fri11] O. Friedrich. „An Integrated, Interactive Application Environment for Session-Oriented IPTV Systems, Enabling Shared User Experiences“. In: (2011) (cit. on p. 14).

[Fuj+15] S. Fujimura, H. Watanabe, A. Nakadaira, T. Yamada, A. Akutsu, and J. J. Kishigami. „BRIGHT: A Concept for a Decentralized Rights Management System based on Blockchain“. In: *2015 IEEE 5th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*. 2015, pp. 345–346 (cit. on p. 28).

[Gar+03] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. „Terra: A Virtual Machine-based Platform for Trusted Computing“. In: *ACM SIGOPS Operating Systems Review*. Vol. 37. 5. ACM. 2003, pp. 193–206 (cit. on p. 28).

[GF04] N. Ghodke and R. Figueiredo. „On the Implications of Machine Virtualization for DRM and Fair Use: A Case Study of a Virtual Audio Device Driver“. In: *Proceedings of the 4th ACM Workshop on Digital Rights Management*. DRM '04. Washington DC, USA: ACM, 2004, pp. 91–98 (cit. on pp. 40, 173–175).

[Ggl] *Widevine DRM Architecture Overview*. Google - Confidential. Google Inc., Mar. 2017 (cit. on p. 7).

[Ghi+13] M. Ghiglieri, F. Oswald, and E. Tews. „HbbTV - I Know What You Are Watching“. In: *13. Deutscher IT-Sicherheitskongress*. SecuMedia Verlags-GmbH, 2013 (cit. on pp. 7, 36, 44).

[GK+10] M. G Kienzle, B. La Gree, and F. Schaffa. „Smarter TV-Linking Broadcast and Broadband Television through a Service Delivery Platform in the Cloud.“ In: *Revista de Radiodifusão-SET* 4.04 (2010).

[GT14] M. Ghiglieri and E. Tews. „A Privacy Protection System for HbbTV in Smart TVs“. English. In: *2014 IEEE 11th Consumer Communications and Networking Conference (CCNC)*. Las Vegas, NV, Jan. 2014, pp. 648–653 (cit. on pp. 7, 36, 44, 176, 177).

[Gud+11] K. Gudeth, M. Pirretti, K. Hoeper, and R. Buskey. „Delivering Secure Applications on Commercial Mobile Devices: The Case for Bare Metal Hypervisors“. In: *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*. SPSM '11. Chicago, Illinois, USA: ACM, 2011, pp. 33–38 (cit. on p. 29).

[HH15] F. M. Hess and M. B. Horn. *Private Enterprise and Public Education*. Teachers College Press, Apr. 2015 (cit. on p. 14).

[Hoe+13] M. Hoekstra, R. Lal, and P. Pappachan. „Using Innovative Instructions to Create Trustworthy Software Solutions“. In: *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy - HASP '13* (2013), pp. 1–8 (cit. on p. 30).

[How+11] C. Howson, E. Gautier, P. Gilberton, A. Laurent, and Y. Legallais. „Combining Future Internet Media with Broadcast TV Content“. In: Citeseer, 2011 (cit. on pp. 43, 176, 177).

[Htm] *HTML5. A Vocabulary and Associated APIs for HTML and XHTML*. W3C, Oct. 2014 (cit. on p. 5).

[Hus+05] W. H. Hussin, P. Coulton, and R. Edwards. „Mobile Ticketing System Employing TrustZone Technology“. In: *International Conference on Mobile Business, 2005. ICMB 2005.* 2005, pp. 651–654 (cit. on pp. 29, 31, 42, 173–175).

[Hwa+08] J. Y. Hwang, S. B. Suh, S. K. Heo, C. J. Park, J. M. Ryu, S. Y. Park, and C. R. Kim. „Xen on ARM: System Virtualization Using Xen Hypervisor for ARM-Based Secure Mobile Phones“. In: *2008 5th IEEE Consumer Communications and Networking Conference*. 2008, pp. 257–261.

[Här+05] H. Härtig, M. Hohmuth, N. Feske, and C. Helmuth. „The Nizza Secure-system Architecture.“ In: *IEEE CollaborateCom 2005, San Jose, CA, USA* (2005) (cit. on p. 42).

[Jen+14] G. Jenkin, M. C. Liassides, J. L. Gilmour, C. G. Hooks, and D. F. Evans. „Virtual Set-Top Box that Executes Service Provider Middleware“. Pat. US Patent 8,683,543. 2014.

[JR11] R. Johnson and M. Rubnich. „Implementing a Key Recovery Attack on the High-bandwidth Digital Content Protection Protocol“. In: *2011 IEEE Consumer Communications and Networking Conference (CCNC)*. 2011, pp. 313–317.

[Js2] *ECMAScript® 2016 Language Specification*. ECMA International, June 2016 (cit. on p. 14).

[Jur+09] A. Jurgelionis, P. Fechteler, P. Eisert, et al. „Platform for Distributed 3D Gaming“. In: *International Journal of Computer Games Technology - Special Issue on Cyber Games and Interactive Entertainment* 2009 (Jan. 2009), 1:1–1:15 (cit. on p. 38).

[Kap+14] A. Kapravelos, C. Grier, N. Chachra, C. Kruegel, G. Vigna, and V. Paxson. „Hulk: Eliciting Malicious Behavior in Browser Extensions“. In: *Proceedings of the 23rd USENIX Conference on Security Symposium*. SEC'14. San Diego, CA: USENIX Association, 2014, pp. 641–654 (cit. on pp. 6, 17).

[KC06] S. T. King and P. M. Chen. „SubVirt: Implementing Malware with Virtual Machines“. In: *2006 IEEE Symposium on Security and Privacy (S P'06)*. 2006, 14 pp.–327 (cit. on pp. 39, 171, 172).

[Kim+97] W.-H. Kim, K.-J. Chen, and H.-S. Cho. „Design and Implementation of MPEG-2/DVB Scrambler Unit and VLSI Chip“. In: *IEEE Transactions on Consumer Electronics* 43.3 (1997), pp. 980–985 (cit. on p. 157).

[Kis+15] J. Kishigami, S. Fujimura, H. Watanabe, A. Nakadaira, and A. Akutsu. „The Blockchain-based Digital Content Distribution System“. In: *2015 IEEE Fifth International Conference on Big Data and Cloud Computing*. 2015, pp. 187–190 (cit. on p. 28).

[Kle+09] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, et al. „seL4: Formal Verification of an OS Kernel“. In: *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*. ACM. 2009, pp. 207–220 (cit. on p. 29).

[Koe13] O. Koemmerling. *Card Sharing Countermeasures*. US Patent App. 13/512,083. 2013.

[Kos+09] K. Kostiainen, J.-E. Ekberg, N. Asokan, and A. Rantala. „On-board Credentials with Open Provisioning“. In: *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*. ASIACCS '09. Sydney, Australia: ACM, 2009, pp. 104–115 (cit. on pp. 29, 31).

[KS96] D. R. Karger and C. Stein. „A New Approach to the Minimum Cut Problem“. In: *J. ACM* 43.4 (July 1996), pp. 601–640 (cit. on p. 38).

[Kuh+98] G. J. Kuhn, D. W. Davies, and S. P. A. Rix. *System and Apparatus for Blockwise Encryption/Decryption of Data*. US Patent 5,799,089. 1998 (cit. on p. 157).

[Lee+12] K. H. Lee, D. H. Kim, and G. T. Baek. „Web Application Virtualization for IPTV“. In: *Network Operations and Management Symposium (APNOMS), 2012 14th Asia-Pacific*. 2012, pp. 1–4 (cit. on pp. 37, 171, 172).

[Lee+13] H. Lee, C. Seo, and S. U. Shin. „DRM Cloud Architecture and Service Scenario for Content Protection“. In: *Journal of Internet Services and Information Security (JISIS)* 3.3/4 (2013), pp. 94–105 (cit. on pp. 40, 44, 173–177).

[LG07] W. Li and D. Gu. „Security Analysis of DVB Common Scrambling Algorithm“. In: *The First International Symposium on Data, Privacy, and E-Commerce, 2007. ISDPE 2007.* 2007, pp. 271–273 (cit. on p. 157).

[Lia+16] X. Liao, L. Lin, G. Tan, H. Jin, X. Yang, W. Zhang, and B. Li. „LiveRender: A Cloud Gaming System Based on Compressed Graphics Streaming“. In: *IEEE/ACM Transactions on Networking* 24.4 (2016), pp. 2128–2139 (cit. on pp. 38, 171, 172).

[Liu+03] Q. Liu, R. Safavi-Naini, and N. P. Sheppard. „Digital Rights Management for Content Distribution“. In: *Proceedings of the Australasian Information Security Workshop Conference on ACSW Frontiers 2003 - Volume 21*. ACSW Frontiers '03. Adelaide, Australia: Australian Computer Society, Inc., 2003, pp. 49–58 (cit. on pp. 160, 161).

[Liv+15] D. Livshits, A. Mikityuk, S. Pham, and A. Shabtai. „Towards Security of Native DRM Execution in HTML5“. In: *2015 IEEE International Symposium on Multimedia (ISM)*. 2015, pp. 411–416 (cit. on p. 10).

[Luk+11] Z. Lukac, M. Radonjic, B. Veris, T. Maruna, and N. Kuzmanovic. „The Experience of Implementing a Hybrid Broadcast Broadband Television on Network-enabled TV Set“. In: *MIPRO, 2011 Proceedings of the 34th International Convention*. 2011, pp. 840–844.

[LW14] Z. Liu and D. S. Wong. *Practical Attribute-Based Encryption: Traitor Tracing, Revocation, and Large Universe*. Cryptology ePrint Archive, Report 2014/616. `http://eprint.iacr.org/2014/616`. 2014 (cit. on p. 26).

[McD91] J. A. McDermid. *Software Engineer's Reference Book*. Elsevier, 1991 (cit. on p. 9).

[McK+13] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar. „Innovative Instructions and Software Model for Isolated Execution“. In: *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*. HASP '13. Tel-Aviv, Israel: ACM, 2013, 10:1–10:1 (cit. on p. 30).

[MF14] A. Mikityuk and O. Friedrich. „Securing Virtual Service Generation on the Network: Adapting Digital Rights Management to Cloud-delivered Media“. In: *2014 ASE BIGDATA/SOCIALCOM/CYBERSECURITY Conference*. 2014 (cit. on p. 153).

[Mic15] B. Michéle. *Smart TV Security: Media Playback and Digital Video Broadcast*. Springer, 2015 (cit. on pp. 7, 36, 37, 84, 121).

[Mik+13] A. Mikityuk, J. P. Seifert, and O. Friedrich. „The Virtual Set-Top Box: On the shift of IPTV Service Execution, Service and UI Composition into the Cloud“. In: *2013 17th International Conference on Intelligence in Next Generation Networks (ICIN)*. 2013, pp. 1–8 (cit. on p. 10).

[Mik+14a] A. Mikityuk, B. Zachey, and O. Friedrich. „Digital Rights Management and its Evolution in the Context of IPTV Platforms in the Web Domain“. In: *2014 IEEE/CIC International Conference on Communications in China (ICCC)*. 2014, pp. 193–198 (cit. on p. 153).

[Mik+14b] A. Mikityuk, J. P. Seifert, and O. Friedrich. „Paradigm Shift in IPTV Service Generation: Comparison between Locally- and Cloud-rendered IPTV UI“. In: *2014 IEEE 11th Consumer Communications and Networking Conference (CCNC)*. 2014, pp. 205–212 (cit. on p. 153).

[Mik+15a] A. Mikityuk, O. Friedrich, R. Skupin, Y. Sanchéz, and T. Schierl. „Compositing without Transcoding for H.265/HEVC in Cloud IPTV and VoD Services“. In: *2015 IEEE 5th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*. 2015, pp. 176–180 (cit. on p. 153).

[Mik+15b] A. Mikityuk, S. Pham, S. Kaiser, O. Friedrich, and S. Arbanowski. „Content Protection in HTML5 TV Platforms: Towards Browser-agnostic DRM and Cloud UI Environments“. In: *Proceedings of the 5th International Workshop on Trustworthy Embedded Devices*. TrustED '15. Denver, Colorado, USA: ACM, 2015, pp. 43–52 (cit. on p. 10).

[Mik+15c] A. Mikityuk, O. Friedrich, and R. Nikutta. „HbbTV Goes Cloud: Decoupling Application Signaling and Application Execution in Hybrid TV“. In: *Proceedings of the ACM International Conference on Interactive Experiences for TV and Online Video*. TVX '15. Brussels, Belgium: ACM, 2015, pp. 191–196 (cit. on p. 10).

[Mik+15d] A. Mikityuk, M. Platschek, and O. Friedrich. „On Virtualization of Red-Button Signaling in Hybrid TV“. In: *2015 IEEE International Symposium on Multimedia (ISM)*. 2015, pp. 435–440 (cit. on p. 10).

[MK14] B. Michéle and A. Karpow. „Watch and be Watched: Compromising All Smart TV Generations“. In: *Consumer Communications and Networking Conference (CCNC), 2014 IEEE*. Jan. 2014, pp. 642–647 (cit. on pp. 7, 44, 176, 177).

[MK90] R. Mori and M. Kawahara. „Superdistribution: The Concept and The Architecture“. In: *IEICE TRANSACTIONS (1976-1990)* 73.7 (1990), pp. 1133–1146 (cit. on p. 28).

[MK97] R. Mori and M. Kawahara. „Superdistribution: An Electronic Infrastructure for the Economy of the Future“. In: 38.7 (1997), pp. 1465–1472 (cit. on p. 28).

[Moh+14] M. Mohanty, V. Do, and C. Gehrmann. „Media Data Protection during Execution on Mobile Platforms – A Review“. In: *SICS Technical Report: T2014:02*. SICS Swedish ICT AB (2014).

[Mor05] S. Morris. *Interactive TV Standards: A Guide to MHP, OCAP, and JavaTV*. Elsevier, San Diego, CA, 2005.

[Mor89] R. Mori. *What Lies Ahead*. Byte Magazine. `http://virtualschool.edu/mon/ElectronicProperty/MoriWhatLiesAhead.html`. 1989 (cit. on p. 28).

[Nga+16] B. Ngabonziza, D. Martin, A. Bailey, H. Cho, and S. Martin. „TrustZone Explained: Architectural Features and Use Cases“. In: *IEEE 2nd International Conference on Collaboration and Internet Computing (CIC), 2016*. IEEE. 2016, pp. 445–451 (cit. on p. 28).

[Nis+06] Y. Nishimoto, A. Baba, and K. Ogawa. „Advanced Conditional Access System for Digital Broadcasting Receivers Using Metadata“. In: *2006 Digest of Technical Papers International Conference on Consumer Electronics*. 2006, pp. 111–112.

[O'L09] E. O'Loughlin. *An Introduction to Business Systems Analysis: Problem Solving Techniques and Strategies*. The Liffey Press, 2009 (cit. on pp. 47, 53).

[OK15] Y. Oren and A. D. Keromytis. „Attacking the Internet Using Broadcast Digital Television“. In: *ACM Transactions on Information and System Security (TISSEC)* 17.4 (Apr. 2015), 16:1–16:27 (cit. on pp. 7, 36).

[Osi17] E. Osiptschuk. „Secure HbbTV for Cloud Browser Platform“. MA thesis. Germany: Technical University Berlin. Chair for Security in Telecommunications., Feb. 2017 (cit. on p. 87).

[Ou+09] W. Ou, X. Wang, W. Han, and Y. Wang. „Research on Trust Evaluation Model Based on TPM“. In: *Fourth International Conference on Frontier of Computer Science and Technology, 2009 (FCST '09)*. 2009, pp. 593–597.

[PB03] S. Pearson and B. Balacheff. *Trusted Computing Platforms: TCPA Technology in Context*. Prentice Hall Professional, 2003 (cit. on p. 30).

[Pet12] R. Petrlic. „Proxy Re-encryption in a Privacy-preserving Cloud Computing DRM Scheme“. In: *Proceedings of the 4th International Conference on Cyberspace Safety and Security*. CSS'12. Melbourne, Australia: Springer-Verlag, 2012, pp. 194–211 (cit. on pp. 40, 173–175).

[Pha+16] S. Pham, A. Mikityuk, J. Barta, and S. Arbanowski. „Commercial OTT Media Delivery Strategy: Streaming Formats and Content Protection“. In: *2016 NAB BEC - OTT Delivery of Video Content*. 2016 (cit. on pp. 10, 15, 23).

[Pin+14] S. Pinto, D. Oliveira, J. Pereira, N. Cardoso, M. Ekpanyapong, J. Cabral, and A. Tavares. „Towards a Lightweight Embedded Virtualization Architecture Exploiting ARM TrustZone“. In: *Emerging Technology and Factory Automation (ETFA), 2014 IEEE*. 2014, pp. 1–4 (cit. on pp. 29, 42).

[Rad15] J. Radhakrishnan. „Hardware Dependency and Performance of JavaScript Engines Used in Popular Browsers“. In: *2015 International Conference on Control Communication Computing India (ICCC)*. 2015, pp. 681–684 (cit. on pp. 1, 15).

[Rei+09] C. Reis, A. Barth, and C. Pizano. „Browser Security: Lessons from Google Chrome“. In: *Queue* 7.5 (June 2009), 3:3–3:8 (cit. on p. 2).

[Ris+09] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. „Hey, You, Get off of My Cloud: Exploring Information Leakage in Third-party Compute Clouds“. In: *Proceedings of the 16th ACM Conference on Computer and Communications Security*. CCS '09. Chicago, Illinois, USA: ACM, 2009, pp. 199–212 (cit. on pp. 39, 171, 172).

[RT08] J. Rutkowska and A. Tereshkin. „Bluepilling the Xen Hypervisor“. In: *Black Hat USA* (2008) (cit. on pp. 39, 171, 172).

[Rus81] J. M. Rushby. *Design and Verification of Secure Systems*. Vol. 15. 5. ACM, 1981 (cit. on p. 29).

[Sab+15a] M. Sabt, M. Achemlal, and A. Bouabdallah. „The Dual-Execution-Environment Approach: Analysis and Comparative Evaluation“. In: *IFIP International Information Security Conference*. Springer. 2015, pp. 557–570 (cit. on pp. 28, 29).

[Sab+15b] M. Sabt, M. Achemlal, and A. Bouabdallah. „Trusted Execution Environment: What It Is, and What It Is Not“. In: *Trustcom/BigDataSE/ISPA, 2015 IEEE*. Vol. 1. IEEE. 2015, pp. 57–64 (cit. on pp. 28, 29).

[San+14] N. Santos, H. Raj, S. Saroiu, and A. Wolman. „Using ARM TrustZone to Build a Trusted Language Runtime for Mobile Applications“. In: *ACM SIGARCH Computer Architecture News* 42.1 (Feb. 2014), pp. 67–80 (cit. on pp. 29, 31, 41, 173–175).

[Sch+11] P. Schoo, V. Fusenig, V. Souza, M. Melo, P. Murray, H. Debar, H. Medhioub, and D. Zeghlache. „Challenges for Cloud Networking Security“. In: *Mobile Networks and Management: Second International ICST Conference, MONAMI 2010, Santander, Spain, September 22-24, 2010, Revised Selected Papers*. Ed. by K. Pentikousis, R. Agüero, M. García-Arranz, and S. Papavassiliou. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 298–313 (cit. on pp. 39, 171, 172).

[Sch95] B. Schneier. *Applied Cryptography (2nd Ed.): Protocols, Algorithms, and Source Code in C*. New York, NY, USA: John Wiley & Sons, Inc., 1995 (cit. on p. 90).

[She15] D. Shen. „Exploiting TrustZone on Android“. In: *Black Hat US* (2015) (cit. on p. 42).

[Son+01] D. X. Song, D. Wagner, and X. Tian. „Timing Analysis of Keystrokes and Timing Attacks on SSH“. In: *Proceedings of the 10th Conference on USENIX Security Symposium - Volume 10*. SSYM'01. Washington, D.C.: USENIX Association, 2001 (cit. on p. 39).

[Sti16] R. Stinder. „Towards Secure Web Content Protection“. MA thesis. Germany: Technical University Berlin. Chair for Security in Telecommunications., Dec. 2016 (cit. on pp. 31, 87, 109, 112).

[Sto11] T. Stockhammer. „Dynamic Adaptive Streaming over HTTP–: Standards and Design Principles“. In: *Proceedings of the Second Annual ACM Conference on Multimedia Systems*. ACM. 2011, pp. 133–144 (cit. on p. 19).

[Suh+03] G. E. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas. „AEGIS: Architecture for Tamper-evident and Tamper-resistant Processing“. In: *Proceedings of the 17th Annual International Conference on Supercomputing*. ICS '03. San Francisco, CA, USA: ACM, 2003, pp. 160–171 (cit. on pp. 29, 41, 173–175).

[Suh+05] G. E. Suh, C. W. O'Donnell, I. Sachdev, and S. Devadas. „Design and Implementation of the AEGIS Single-chip Secure Processor Using Physical Random Functions“. In: *32nd International Symposium on Computer Architecture (ISCA'05)*. 2005, pp. 25–36 (cit. on pp. 29, 41).

[Suk+15] K. Suksomboon, M. Fukushima, and M. Hayashi. „Optimal Virtualization of Functionality for Customer Premise Equipment“. In: *2015 IEEE International Conference on Communications (ICC)*. 2015, pp. 5685–5690 (cit. on pp. 38, 60, 171, 172).

[Svg] *Scalable Vector Graphics (SVG*. W3C, Aug. 2011 (cit. on p. 16).

[Tew+12] E. Tews, J. Wälde, and M. Weiner. „Breaking DVB-CSA“. In: *Proceedings of the 4th Western European Conference on Research in Cryptology*. WEWoRC'11. Weimar, Germany: Springer-Verlag, 2012, pp. 45–61.

[Tha+15a] D. Thatmann, A. Butyrtschik, and A. Küpper. „A Secure DHT-based Key Distribution System for Attribute-based Encryption and Decryption“. In: *Proceedings of the 9th Intl. Conference on Signal Processing and Communication Systems (ICSPCS 2015)*. Cairns, Australia: IEEE, 2015 (cit. on p. 26).

[Tha+15b] D. Thatmann, S. Zickau, A. Förster, and A. Küpper. „Applying Attribute-based Encryption on Publish Subscribe Messaging Patterns for the Internet of Things“. In: *Proceedings of the 8th IEEE International. Conference on Internet of Things (iThings 2015)*. Sydney, Australia: IEEE, 2015.

[Tha+16] D. Thatmann, P. Raschke, and A. Küpper. „"Please, no more GUIs!": A User Study, Prototype Development and Evaluation on the Integration of Attribute-based Encryption in a Hospital Environment“. In: *40th IEEE Computer Society International Conference on Computers, Software and Applications (Compsac 2016) - User Centered Design and Adaptive Systems (UCDAS)*. Atlanta, USA: IEEE Computer Society, 2016.

[The+00] D. L. C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell, and M. Horowitz. „Architectural Support for Copy and Tamper Resistant Software“. In: *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS IX. Cambridge, Massachusetts, USA: ACM, 2000, pp. 168–177 (cit. on pp. 41, 173–175).

[The+13] M. Theoharidou, N. Tsalis, and D. Gritzalis. „In Cloud We Trust: Risk-Assessment-as-a-Service“. In: *IFIP International Conference on Trust Management*. Springer. 2013, pp. 100–110.

[Tsa+14] N. Tsalis, N. Virvilis, A. Mylonas, T. Apostolopoulos, and D. Gritzalis. „Browser Blacklists: The Utopia of Phishing Protection“. In: *International Conference on E-Business and Telecommunications*. Springer. 2014, pp. 278–293 (cit. on p. 17).

[Tsa+15] N. Tsalis, A. Mylonas, and D. Gritzalis. „An Intensive Analysis of Security and Privacy Browser Add-ons“. In: *International Conference on Risks and Security of Internet and Systems*. Springer. 2015, pp. 258–273 (cit. on p. 17).

[Vas+14] A. Vasudevan, J. M. McCune, and J. Newsome. *Trustworthy Execution on Mobile Devices*. Springer, 2014 (cit. on p. 28).

[Vil+12] Á. Villegas, P. Pérez, J. María Cubero, E. Estalayo, and N. García. „Network Assisted Content Protection Architectures for a Connected World“. In: *Bell Labs Technical Journal* 16.4 (Mar. 2012), pp. 85–96 (cit. on pp. 40, 173–175).

[Vir+14] N. Virvilis, N. Tsalis, A. Mylonas, and D. Gritzalis. „Mobile Devices: A Phisher's Paradise“. In: *2014 11th International Conference on Security and Cryptography (SECRYPT)*. 2014, pp. 1–9 (cit. on p. 17).

[Vir+15] N. Virvilis, A. Mylonas, N. Tsalis, and D. Gritzalis. „Security Busters: Web Browser Security vs. Rogue Sites“. In: *Computers & Security* 52 (2015), pp. 90–105 (cit. on p. 17).

[Wei+16] N. Weichbrodt, A. Kurmus, P. Pietzuch, and R. Kapitza. „AsyncShock: Exploiting Synchronisation Bugs in Intel SGX Enclaves“. In: *21st European Symposium on Research in Somputer Science (ESORICS 2016)* (2016) (cit. on p. 43).

[Wie99] K. Wiegers. „First Things First: Prioritizing Requirements“. In: *Software Development* 7.9 (1999), pp. 48–53 (cit. on p. 53).

[Wil+07] P. Wilson, A. Frey, T. Mihm, D. Kershaw, and T. Alves. „Implementing Embedded Security on Dual-Virtual-CPU Systems“. In: *IEEE Des. Test* 24.6 (Nov. 2007), pp. 582–591 (cit. on pp. 29, 30).

[Wir05] K. Wirt. „Fault Attack on the DVB Common Scrambling Algorithm“. In: *Computational Science and Its Applications – ICCSA 2005: International Conference, Singapore, May 9-12, 2005, Proceedings, Part II*. Ed. by O. Gervasi, M. L. Gavrilova, V. Kumar, A. Laganà, H. P. Lee, Y. Mun, D. Taniar, and C. J. K. Tan. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 577–584 (cit. on p. 157).

[WR09] R. Wojtczuk and J. Rutkowska. „Attacking Intel® Trusted Execution Technology“. In: *Black Hat DC*. Invisible Things Lab (2009) (cit. on p. 43).

[WW05] R.-P. Weinmann and K. Wirt. „Analysis of the DVB Common Scrambling Algorithm“. In: *Communications and Multimedia Security*. Springer. 2005, pp. 195–207 (cit. on p. 159).

[Xia+07] Y. Xiao, X. Du, and J. Zhang. „Internet Protocol Television (IPTV): The Killer Application for the Next-Generation Internet“. In: *Institute of Electrical and Electronics Engineers*. 2007 (cit. on p. 160).

[Zha+07] X. Zhang, O. Aciicmez, and J.-P. Seifert. „A Trusted Mobile Phone Reference Architecture via Secure Kernel“. In: *Proceedings of the 2007 ACM Workshop on Scalable Trusted Computing*. STC '07. Alexandria, Virginia, USA: ACM, 2007, pp. 7–14 (cit. on p. 30).

[Zha+16] N. Zhang, K. Sun, D. Shands, W. Lou, and Y. T. Hou. „TruSpy: Cache Side-Channel Information Leakage from the Secure World on ARM Devices“. In: *IACR Cryptology ePrint Archive* 2016 (Oct. 2016), p. 980 (cit. on p. 42).

[Zho+12] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang. „Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets.“ In: *NDSS*. Vol. 25. 4. 2012, pp. 50–52 (cit. on p. 3).

[Zhu+11] M. Zhu, S. Mondet, G. Morin, W. T. Ooi, and W. Cheng. „Towards Peer-assisted Rendering in Networked Virtual Environments“. In: *Proceedings of the 19th ACM International Conference on Multimedia*. MM '11. Scottsdale, Arizona, USA: ACM, 2011, pp. 183–192 (cit. on pp. 37, 171, 172).

[Zou+10] P. Zou, C. Wang, Z. Liu, and D. Bao. „Phosphor: A Cloud-based DRM Scheme with Sim Card“. In: *Web Conference (APWEB), 2010 12th International Asia-Pacific*. 2010, pp. 459–463 (cit. on pp. 40, 173–175).

[A. 16] A. Mikityuk et al. *Cloud Browser Business Opportunities*. GSMA, 2016 (cit. on p. 121).

[ARM09b] ARM Limited. *ARM1176JZF-S. Technical Reference Manual. Revision: r0p7.* 2009 (cit. on p. 31).

[ARM13] ARM Holdings plc. *Strategic Report*. 2013 (cit. on p. 30).

[ARM14] ARM. *ARM® Cortex®-A53 MPCore Processor*. Technical Reference Manual. 2014 (cit. on p. 112).

[ATS11] ATSC Planning Team 2. *Final Report on ATSC 3.0 Next Generation Broadcast Television*. PT2-046r11. Washington, D.C., USA: ATSC, Sept. 2011.

[Adv05] Advanced Micro Devices, Inc. *Secure Virtual Machine Architecture*. May 2005.

[C. 17] C. Meerveld and A. Mikityuk. *Cloud Browser Architecture*. W3C Interest Group, June 2017 (cit. on p. 66).

[CEA11] CEA-2014. *Web-based Protocol and Framework for Remote User Interface on UPnP Networks and the Internet (Web4CE)*. Standard. CEA, Jan. 2011 (cit. on p. 164).

[CI 16] CI Plus. *CI Plus Specification - Content Security Extensions to the Common Interface*. Technical Specification. Guildford, UK: CI Plus LLP, May 2016 (cit. on p. 159).

[Con16b] Consumer Technology Association. *Web Application Video Ecosystem (WAVE): Scope of Work*. CTA, Sept. 2016 (cit. on pp. 14, 27).

[DVB96] DVB. *DVB Common Scrambling Algorithm - Distribution Agreements*. June 1996 (cit. on p. 157).

[Doh82] Doherty, W. J., and Thadhani, A. J. *The Economic Value of Rapid Response Time.* IBM, 1982 (cit. on p. 120).

[ETS08a] ETSI. *Digital Video Broadcasting (DVB); DVB Specification for Data Broadcasting*. European Standard (Telecommunications series) EN 301 192. Sophia Antipolis, France: ETSI, Apr. 2008 (cit. on p. 34).

[ETS08b] ETSI. *Digital Video Broadcasting (DVB); Head-end Implementation of DVB SimulCrypt*. ETSI Standard TS 103 197. Sophia Antipolis, France: ETSI, Oct. 2008 (cit. on p. 159).

[ETS09a] ETSI. *Digital Video Broadcasting (DVB); Framing Structure, Channel Coding and Modulation for Digital Terrestrial Television*. European Standard (Telecommunications series) EN 300 744. Sophia Antipolis, France: ETSI, Jan. 2009 (cit. on p. 165).

[ETS09b] ETSI. *Digital Video Broadcasting (DVB); Transport of MPEG-2 TS Based DVB Services over IP Based Networks*. Technical Specification TS 102 034. Sophia Antipolis, France: ETSI, Aug. 2009 (cit. on p. 165).

[ETS10a] ETSI. *Access, Terminals, Transmission and Multiplexing (ATTM); Integrated Broadband Cable and Television Networks; K-LAD Functional Specification*. Technical Specification TS 103 162. Sophia Antipolis, France: ETSI, Oct. 2010 (cit. on p. 160).

[ETS10b] ETSI. *Digital Video Broadcasting (DVB); Multimedia Home Platform (MHP)*. Technical Specification TS 102 727. Sophia Antipolis, France: ETSI, Jan. 2010 (cit. on pp. 7, 32, 95).

[ETS10c] ETSI. *Digital Video Broadcasting (DVB); Signalling and Carriage of Interactive Applications and Services in Hybrid Broadcast/Broadband Environments*. Technical Specification TS 102 809. Sophia Antipolis, France: ETSI, Jan. 2010 (cit. on pp. 164, 165).

[ETS10d] ETSI. *Hybrid Broadcast Broadband TV*. Technical Specification TS 102 796. Sophia Antipolis, France: ETSI, June 2010 (cit. on p. 32).

[ETS10e] ETSI. *MHEG-5 Broadcast Profile*. Standard ES 202 184. Sophia Antipolis, France: ETSI, Jan. 2010 (cit. on p. 32).

[ETS11] ETSI. *Digital Video Broadcasting (DVB); Support for Use of the DVB Scrambling Algorithm Version 3 Within Digital Broadcasting Systems*. Technical Specification TS 100 289. Sophia Antipolis, France: ETSI, Sept. 2011 (cit. on p. 157).

[ETS12] ETSI. *Hybrid Broadcast Broadband TV (DVB); Technical Specification*. Technical Specification TS 102 796. Sophia Antipolis, France: ETSI, Nov. 2012 (cit. on p. 164).

[ETS13] ETSI. *Digital Video Broadcasting (DVB); Signalling and Carriage of Interactive Applications and Services in Hybrid Broadcast/Broadband Environments*. Technical Specification TS 102 809. Sophia Antipolis, France: ETSI, July 2013 (cit. on pp. 33, 34).

[ETS15] ETSI. *Hybrid Broadcast Broadband TV*. Technical Specification TS 102 796. Sophia Antipolis, France: ETSI, Oct. 2015 (cit. on pp. 32–34, 165).

[ETS16] ETSI. *Hybrid Broadcast Broadband TV*. Technical Specification TS 102 796. Sophia Antipolis, France: ETSI, Aug. 2016 (cit. on pp. 84, 116, 119, 123).

[ETS96] ETSI. *Digital Video Broadcasting (DVB); Support for Use of Scrambling and Conditional Access (CA) within Digital Broadcasting Systems*. Sophia Antipolis, France: ETSI, Oct. 1996 (cit. on p. 157).

[ETS97] ETSI. *Digital Video Broadcasting (DVB); Framing Structure, Channel Coding and Modulation for 11/12 GHz Satellite Services*. European Standard (Telecommunications series) EN 300 421. Sophia Antipolis, France: ETSI, Aug. 1997 (cit. on p. 165).

[ETS98] ETSI. *Digital Video Broadcasting (DVB); Framing Structure, Channel Coding and Modulation for Cable Systems*. European Standard (Telecommunications series) EN 300 429. Sophia Antipolis, France: ETSI, Apr. 1998 (cit. on p. 165).

[ETS99] ETSI. *Digital Video Broadcasting (DVB); Extensions to the Common Interface Specification*. ETSI Standard TS 101 699. Sophia Antipolis, France: ETSI, Nov. 1999 (cit. on p. 159).

[Fra14a] Fraunhofer Fokus. *Open-source CDM Implementation*. 2014 (cit. on p. 91).

[Fra14b] Fraunhofer Fokus. *Open-source CDMi Implementation*. 2014 (cit. on p. 91).

[Glo17c] GlobalPlatform. *GlobalPlatform Device Technology TEE System Architecture*. Redwood City, CA, USA: GlobalPlatform, Jan. 2017 (cit. on p. 28).

[ISO11] ISO/IEC. *Information Technology – Programming Languages – C*. 9899:2011. ISO, Dec. 2011 (cit. on p. 87).

[ISO12a] ISO/IEC. *Information Technology - Dynamic Adaptive Streaming over HTTP (DASH) - Part 1: Media Presentation Description and Segment Formats*. 23009-1:2012. Geneva, Switzerland: ISO/IEC, Jan. 2012 (cit. on pp. 6, 19, 164).

[ISO12b] ISO/IEC. *Information Technology - MPEG Systems Technologies - Part 7: Common Encryption in ISO Base Media File Format Files*. 23001-7:2012. Geneva, Switzerland: ISO/IEC, Jan. 2012 (cit. on p. 164).

[ISO15] ISO/IEC. *Information Technology – Coding of Audio-visual Objects – Part 12: ISO Base Media File Format*. 14496-12:2015. Geneva, Switzerland: ISO/IEC, Dec. 2015 (cit. on p. 19).

[ISO16] ISO/IEC. *Information Technology – MPEG Systems Technologies – Part 7: Common Encryption in ISO Base Media File Format Files*. International Standard 23001-7:2016. ISO/IEC, Feb. 2016 (cit. on pp. 24, 162).

[ISO94] ISO/IEC. *Information Technology - Generic Coding of Moving Pictures and Associated Audio Information. Part 1: Systems*. International Standard 13818-1:1994. ISO/IEC, Nov. 1994 (cit. on p. 34).

[J. 02] J. Ritchie and T. Kuehnel. *UPnP AV Architecture:1*. Design Document. UPnP, June 2002.

[J. 14a] J. C. Simmons. *Content Decryption Module Interface Specification. An Open Interface for Enabling HTML5 Encrypted Media Extensions in Open Source Browsers.* Microsoft Corporation, Jan. 2014 (cit. on pp. 24, 91).

[J. 14b] J. C. Simmons, Dr. S. Arbanowski. *Interoperability, Digital Rights Management and the Web.* Microsoft Corporation, Fraunhofer FOKUS Research Institute, Sept. 2014 (cit. on pp. 24, 91).

[J. 16] J. Broughton. *Content Owners Look for Weapons to Combat the Digital Pirates*. IHS Technology Insight Report. Broadband Media Intelligence Service, Apr. 2016 (cit. on pp. 2, 3).

[LeF71] LeFevre, R. *The Philosophy of Ownership*. May 1971, pp. 1–94 (cit. on pp. 25, 26).

[Mot15] Motion Picture Laboratories, Inc. *MovieLabs Specification for Enhanced Content Protection*. San Francisco, CA, USA: MovieLabs, Feb. 2015 (cit. on pp. 7, 24, 28).

[OIP13] OIPF. *Volume 2 - Media Formats*. Release 2 Specification. Sophia Antipolis, France: Open IPTV Forum, May 2013.

[OIP14] OIPF. *Volume 5 - Declarative Application Environment*. Release 2 Specification. Sophia Antipolis, France: Open IPTV Forum, Jan. 2014 (cit. on pp. 95, 116, 119).

[Ope09] Open Mobile Terminal Platform. *Advanced Trusted Environment*. OMPT Limited, May 2009 (cit. on p. 28).

[S. 97] S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels*. IETF Network Working Group, Mar. 1997 (cit. on p. 53).

[W3C16] W3C. *Encrypted Media Extensions*. Ed. by D. Dorwin, J. Smith, and M. Watson. W3C, Feb. 2016 (cit. on p. 91).

Webseiten

[Fou] Free Software Foundation. *Defective by Design*. URL: `https://www.defectivebydesign.org/` (visited on Dec. 15, 2016) (cit. on p. 26).

[Gal] A. Gal. *Reconciling Mozilla's Mission and W3C EME*. URL: `https://hacks.mozilla.org/2014/05/reconciling-mozillas-mission-and-w3c-eme/` (visited on Nov. 21, 2016).

[M4V] M4VGear. *Why We Need To Remove DRM*. URL: `http://www.m4vgear.com/guide/why-we-need-to-remove-drm.html` (visited on Dec. 15, 2016) (cit. on p. 26).

[Man] J. T. K. Man. *Broadcast Encryption*. URL: `http://www.cs.uu.nl/docs/vakken/b3sec/Proj12/Broadcast.pdf` (visited on Oct. 31, 2016) (cit. on p. 159).

[Pla] L. Plaugic. *Spotify's Year in Music Shows Just How Little We Pay Artists for Their Music*. URL: http://www.theverge.com/2015/12/7/9861372/spotify-year-in-review-artist-payment-royalties (visited on Dec. 15, 2016) (cit. on p. 27).

[tec] techcrunch.com. *Say Goodbye to NPAPI*. URL: https://techcrunch.com/2013/09/23/say-goodbye-to-npapi/ (visited on Nov. 15, 2016) (cit. on p. 163).

[the] thenextweb.com. *Google Looks to Drop Netscape Plugin API Support in Chrome*. URL: http://thenextweb.com/google/2013/09/23/google-chrome-drops-netscape-plugin-api-support-to-improve-stability-will-block-most-plugins-in-january-2014/ (visited on Nov. 15, 2016) (cit. on p. 163).

[Voo] B. Voogt. *Understand Publishing*. URL: http://www.digitalmusicnews.com/2014/02/28/understandpublishing/ (visited on Dec. 15, 2016) (cit. on p. 26).

[W3C] W3C. *Web and TV Interest Group*. URL: https://www.w3.org/2011/webtv/ (visited on Dec. 17, 2016) (cit. on pp. 5, 27).

[96B16a] 96Boards. *96Boards: HiKey Board Debian OS*. 2016. URL: https://www.96boards.org/documentation/ConsumerEdition/HiKey/Downloads/Debian.md/ (visited on June 10, 2017) (cit. on pp. 92, 95).

[96B16b] 96Boards. *HiKey Board*. 2016. URL: http://www.96boards.org/product/hikey/ (visited on June 14, 2017) (cit. on p. 112).

[A. 04] A. Lau. *A Look at How Device Fragmentation Influences J2ME Application Development*. 2004. URL: http://www.javaworld.com/article/2072740/mobile-java/the-fragmentation-effect.html (visited on May 25, 2017) (cit. on p. 1).

[A. 14] A. Pennington. *The Stick Gets Smart*. 2014. (Visited on July 25, 2015) (cit. on p. 2).

[A. 17] A. Gal. *Reconciling Mozilla's Mission and W3C EME*. 2017. URL: https://hacks.mozilla.org/2014/05/reconciling-mozillas-mission-and-w3c-eme/ (visited on May 24, 2017) (cit. on pp. 7, 24).

[ANS16] ANSOL. *ANSOL, AEL to Lead Demonstration Against Digital Handcuffs at World Wide Web Consortium Meeting*. 2016. URL: https://ansol.org/DRM-no-HTML-EN (visited on Dec. 17, 2016) (cit. on p. 26).

[ARM09a] ARM Limited. *ARM Security Technology. Building a Secure System using TrustZone® Technology*. 2009. URL: http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf (visited on May 29, 2017) (cit. on p. 30).

[ARM17a] ARM. *ARM TrustZone*. 2017. URL: https://www.arm.com/products/security-on-arm/trustzone (visited on May 25, 2017) (cit. on pp. 11, 31, 124).

[ARM17b] ARM Ltd. 2017. URL: https://www.arm.com (visited on May 28, 2017).

[Act] ActiveVideo. URL: http://www.activevideo.com (visited on June 13, 2017) (cit. on p. 127).

[Ado15] Adobe. *Adobe Roadmap For the Flash Runtimes*. 2015. URL: http://wwwimages.adobe.com/content/dam/Adobe/en/devnet/flashplatform/whitepapers/flash-runtimes-roadmap.pdf (visited on Oct. 24, 2016) (cit. on p. 20).

[Ado17a] Adobe Systems Inc. *Access DRM*. 2017. URL: http://www.adobe.com/in/products/adobe-access.html/ (visited on May 28, 2017) (cit. on p. 23).

[Ado17b] Adobe Systems Inc. *HTTP Dynamic Streaming (HDS)*. 2017. URL: http://www.adobe.com/products/hds-dynamic-streaming.html (visited on May 28, 2017) (cit. on p. 18).

[Ado17c] Adobe Systems Incorporated. *Flash Player*. 2017. URL: http://www.adobe.com/products/flashplayer.html (visited on May 28, 2017) (cit. on p. 15).

[Adv16] Advanced Television Systems Committee. 2016. URL: http://www.atsc.org/cms/ (visited on Feb. 12, 2016) (cit. on p. 32).

[Ama17] Amazon.com, Inc. *Amazon.com: Amazon Video*. 2017. URL: https://www.amazon.com/Prime-Video/b?node=2676882011 (visited on May 22, 2017) (cit. on p. 1).

[And17a] Android Developers. *Android NDK*. 2017. URL: https://developer.android.com/ndk/index.html (visited on May 28, 2017) (cit. on p. 155).

[And17b] Android Developers. *Android Studio*. 2017. URL: https://developer.android.com/studio/index.html (visited on May 28, 2017) (cit. on p. 155).

[Apa17] Apache Software Foundation. *Cordova*. 2017. URL: https://cordova.apache.org/ (visited on May 28, 2017) (cit. on pp. 14, 127).

[App17a] Apple Inc. *CoreGraphics*. 2017. URL: https://developers.google.com/v8/ (visited on May 28, 2017) (cit. on p. 17).

[App17b] Apple Inc. *Fairplay DRM*. 2017. URL: https://developer.apple.com/streaming/fps/ (visited on May 28, 2017) (cit. on p. 23).

[App17c] Apple Inc. *HTTP Live Streaming*. 2017. URL: https://developer.apple.com/streaming/ (visited on May 25, 2017) (cit. on p. 6).

[App17d] Apple Inc. *JavaScriptCore*. 2017. URL: https://trac.Webkit.org/wiki/JavaScriptCore (visited on May 28, 2017) (cit. on p. 16).

[App17e] Apple Inc. *Safari*. 2017. URL: https://www.apple.com/lae/safari/ (visited on May 28, 2017) (cit. on p. 15).

[App17f] Apple Inc. *WebKit*. 2017. URL: https://webkit.org/ (visited on May 28, 2017) (cit. on p. 16).

[B. 01] B. Schneier. *The Futility of Digital Copy Prevention*. 2001. URL: http://cryptome.org/futile-cp.htm (visited on Oct. 31, 2016) (cit. on pp. 21, 25).

[Bar08] Baror, Y. and Yogev, A. and Sharabani, A. *Flash Parameter Injection*. 2008. URL: http://blog.watchfire.com/FPI.pdf (visited on Oct. 18, 2016) (cit. on p. 19).

[C. 14] C. Cheevers, CTO Customer Premise Equipment Group in ARRIS. *Changing the role of STB*. 2014. URL: http://cablecongress.com/wp-content/uploads/cablecongress/2014/03/Cheevers-Charles.pdf (visited on July 25, 2015) (cit. on p. 2).

[CIS14] CISCO, VNI. *Cisco Visual Networking Index: Forecast and Methodology, 2013–2018: Visual Networking Index (VNI)*. 2014. URL: http://www.cisco.com/c/dam/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.pdf (visited on Oct. 18, 2016) (cit. on p. 1).

[CTA17] CTA. *Consumer Technology Association*. 2017. URL: https://www.cta.tech/ (visited on May 22, 2017) (cit. on p. 1).

[Clo17] Cloud Browser TF - Web and TV IG. 2017. URL: https://www.telekom.de/ (visited on May 27, 2017) (cit. on pp. 5, 47).

[Con16a] Consumer Electronics Association (CEA). 2016. URL: https://www.ce.org (visited on Feb. 8, 2016) (cit. on p. 164).

[D. 08] D. C. Rajapakse. *Fragmentation of Mobile Applications*. 2008. URL: http://www.comp.nus.edu.sg/~damithch/df/device-fragmentation.htm (visited on May 25, 2017) (cit. on p. 2).

[DAS16a] DASH-IF. *DASH Industry Forum*. 2016. URL: http://dashif.org/ (visited on Oct. 24, 2016) (cit. on p. 20).

[DAS16b] DASH Industry Forum. *A Reference Client Implementation for the Playback of MPEG DASH via Javascript and Compliant Browsers*. 2016. URL: https://github.com/Dash-Industry-Forum/dash.js/ (visited on Oct. 24, 2016) (cit. on p. 20).

[Deu17] Deutsche Telekom. 2017. URL: https://www.telekom.de/ (visited on May 27, 2017) (cit. on p. 47).

[Dig16a] Digital Living Network Alliance. *Informations about the Technology Standards Organization DLNA*. 2016. URL: http://www.dlna.org (visited on Aug. 16, 2016) (cit. on p. 38).

[Dig16b] Digital Video Broadcasting (DVB). 2016. URL: https://www.dvb.org (visited on Feb. 8, 2016) (cit. on pp. 33, 157, 164).

[Dig17] Digital Millennium Copyright Act Services Ltd. 2017. URL: http://www.dmca.com/ (visited on May 28, 2017) (cit. on p. 24).

[Dun] Dune HD. *Dune HD TV-102*. URL: http://dune-hd.com/eng/products/set_top_boxes/34 (visited on June 13, 2017) (cit. on p. 101).

[EUR14] EUROMEDIA Magazine. *STB and Home Gateway Survey*. 2014. URL: http://irdeto.com/documents/Articles/art_euromedia-magazine_JulAug_2014.pdf (visited on July 25, 2015) (cit. on p. 2).

[FFm17] FFmpeg Team. *FFmpeg*. 2017. URL: https://ffmpeg.org/ (visited on May 25, 2017) (cit. on p. 7).

[Fac17] Facebook, Inc. *Facebook*. 2017. URL: https://www.facebook.com/ (visited on May 22, 2017) (cit. on p. 1).

[GSt17] GStreamer. *GStreamer. GStreamer: Open Source Multimedia Framework*. 2017. URL: https://gstreamer.freedesktop.org (visited on June 10, 2017) (cit. on p. 89).

[Git17] GitHub Inc. *GitHub*. 2017. URL: https://github.com/ (visited on May 28, 2017) (cit. on p. 16).

[Glo17a] GlobalPlatform. 2017. URL: https://www.globalplatform.org/ (visited on Apr. 18, 2017) (cit. on p. 28).

[Glo17b] GlobalPlatform. *Device Specifications - Trusted Execution Environment (TEE)*. 2017. URL: http://www.globalplatform.org/specificationsdevice.asp (visited on Apr. 18, 2017) (cit. on pp. 28, 90).

[Goo17a] Google. *The Go Programming Language*. 2017. URL: https://golang.org (visited on June 10, 2017) (cit. on p. 87).

[Goo17b] Google Inc. *Android OS*. 2017. URL: https://www.android.com/ (visited on May 22, 2017) (cit. on pp. 2, 14).

[Goo17c] Google Inc. *Android TV*. 2017. URL: https://www.android.com/tv/ (visited on May 22, 2017) (cit. on p. 2).

[Goo17d] Google Inc. *Angular JS*. 2017. URL: https://angularjs.org/ (visited on May 28, 2017) (cit. on p. 15).

[Goo17e] Google Inc. *Blink*. 2017. URL: https://www.chromium.org/blink (visited on May 28, 2017) (cit. on p. 16).

[Goo17f] Google Inc. *Google Chrome*. 2017. URL: https://www.google.com/chrome/ (visited on May 25, 2017) (cit. on p. 10).

[Goo17g] Google Inc. *Google Chromecast*. 2017. URL: https://www.google.com/chromecast/ (visited on May 22, 2017) (cit. on p. 2).

[Goo17h] Google Inc. *Pepper Plugin API*. 2017. URL: https://code.google.com/p/ppapi/ (visited on May 28, 2017).

[Goo17i] Google Inc. *V8*. 2017. URL: https://developers.google.com/v8/ (visited on May 28, 2017) (cit. on p. 16).

[Goo17j] Google Inc. *Widevine DRM*. 2017. URL: https://www.widevine.com/wv_drm.html (visited on May 28, 2017) (cit. on p. 22).

[Goo17k] Google Inc. Android Developers. *Platform Versions*. 2017. URL: https://developer.android.com/about/dashboards/index.html (visited on May 28, 2017) (cit. on p. 155).

[Hbb] HbbTV. *HbbTV Association Announces Development of Operator Application Specification*. URL: https://www.hbbtv.org/news-events/hbbtv-association-announces-development-of-operator-application-specification/ (visited on June 11, 2017) (cit. on p. 119).

[Hbb15] HbbTV. *HbbTV and Security*. 2015. URL: http://www.hbbtv.org/wp-content/uploads/2015/09/HbbTv-Security-2015.pdf (visited on Dec. 27, 2016) (cit. on pp. 37, 84, 121).

[Hyb16] Hybrid Broadcast Broadband TV (HbbTV). 2016. URL: https://www.hbbtv.org (visited on Jan. 21, 2016) (cit. on pp. 1, 32, 43, 160).

[ITU17] ITU-T. *International Telecommunication Union Telecommunication Standardization Sector*. 2017. URL: http://www.itu.int/ (visited on May 22, 2017) (cit. on p. 1).

[Ima17] Imagination Technologies Limited. 2017. URL: https://www.imgtec.com/mips/ (visited on May 28, 2017).

[Ins17] Instagram, Inc. *Instagram*. 2017. URL: https://www.instagram.com/ (visited on May 22, 2017) (cit. on p. 1).

[Int16] Integrated Services Digital Broadcasting - Terrestrial. 2016. URL: http://www.dibeg.org/ (visited on Feb. 12, 2016) (cit. on p. 32).

[Int17] Intel Corporation. *Intel® Software Guard Extensions (Intel® SGX) SDK*. 2017. URL: https://software.intel.com/en-us/sgx-sdk (visited on May 25, 2017) (cit. on p. 11).

[J. 15a] J. Piesing. *Protecting TV Receivers Against Attacks via the Broadcast*. 2015. URL: http://www.hbbtv.org/wp-content/uploads/2015/12/24_JonPiesing_HbbTV_Presentation.pdf (visited on Dec. 27, 2016) (cit. on p. 37).

[J. 15b] J. Song. *The DTMB Deployment Status in China and the Work for Future*. 2015. URL: https://www.itu.int/en/ITU-R/GE06-Symposium-2015/Session2/202%20%20ITU-ADS%20Symposium-JianSong.pdf (visited on Feb. 12, 2016) (cit. on p. 32).

[J. 17] J. Greene. *Intel® Trusted Execution Technology: White Paper*. 2017. URL: http://www.intel.com/content/www/us/en/architecture-and-technology/trusted-execution-technology/trusted-execution-technology-security-paper.html (visited on May 29, 2017) (cit. on p. 30).

[K. 10] K. Merkel. *HbbTV - A Hybrid Broadcast-broadband System for the Living Room*. Feb. 2010. URL: https://tech.ebu.ch/docs/techreview/trev_2010-Q1_HbbTV.pdf (visited on June 6, 2017).

[K. 16] K. Zetter. *A Bug in Chrome Makes It Easy to Pirate Movies*. June 2016. URL: https://www.wired.com/2016/06/bug-chrome-makes-easy-pirate-movies/ (visited on May 28, 2017) (cit. on pp. 7, 24).

[K. 17] K. Pflug. *What's New in Microsoft Edge in the Windows 10 Creators Update*. Apr. 2017. URL: https://blogs.windows.com/msedgedev/2017/04/11/introducing-edgehtml-15/#jkRXS9475j8Z0zBh.97 (visited on May 28, 2017) (cit. on p. 16).

[Khr17] Khronos Group. *OpenGL*. 2017. URL: https://www.opengl.org/ (visited on May 28, 2017) (cit. on p. 155).

[Lin16] Linaro. *96Boards*. 2016. URL: https://www.96boards.org/ (visited on June 10, 2017) (cit. on p. 90).

[Lin17a] Linaro. 2017. URL: https://www.linaro.org/ (visited on May 29, 2017).

[Lin17b] Linaro. *OP-TEE Trusted OS*. 2017. URL: https://github.com/OP-TEE/optee_os (visited on June 10, 2017) (cit. on p. 90).

[Lin17c] Linaro. *Open Portable Trusted Execution Environment (OP-TEE)*. 2017. URL: https://www.linaro.org/initiatives/op-tee/ (visited on May 29, 2017) (cit. on p. 31).

[Lin17d] Linux KVM. Open Virtualization Alliance (OVA). *Kernel-based Virtual Machine*. 2017. URL: https://www.linux-kvm.org/ (visited on May 29, 2017).

[M. 13] M. Swider. *Apple TV vs Chromecast: Which is Better? Comparison of the Google and Apple Set-Top Boxes*. 2013. URL: http://www.techradar.com/news/television/tv/apple-tv-vs-chromecast-which-is-better--1185491 (visited on July 25, 2015) (cit. on p. 2).

[MHE16] MHEG ISO Multimedia and Hypermedia information coding Expert Group. 2016. URL: http://www.mheg.org/users/mheg/index.php (visited on Jan. 21, 2016) (cit. on p. 32).

[Man17] ManifoldJS. *Manifold JS*. 2017. URL: http://manifoldjs.com/ (visited on May 28, 2017) (cit. on pp. 14, 127).

[Mic17a] Microsoft Corporation. *Microsoft Edge*. 2017. URL: https://www.microsoft.com/en-us/windows/microsoft-edge (visited on May 25, 2017) (cit. on p. 10).

[Mic17b] Microsoft Corporation. *Microsoft PlayReady Enhanced Content Protection*. 2017. URL: https://www.microsoft.com/playready/features/EnhancedContentProtection.aspx (visited on May 24, 2017) (cit. on p. 7).

[Mic17c] Microsoft Corporation. *PlayReady DRM*. 2017. URL: https://www.microsoft.com/playready/ (visited on May 28, 2017) (cit. on p. 22).

[Mic17d] Microsoft Corporation. *Security Levels*. 2017. URL: https://msdn.microsoft.com/en-us/library/windows/desktop/bb649434(v=vs.85).aspx (visited on May 24, 2017) (cit. on p. 7).

[Mic17e] Microsoft Corporation. *Silverlight*. 2017. URL: https://www.microsoft.com/silverlight/ (visited on May 28, 2017) (cit. on p. 15).

[Mic17f] Microsoft Corporation. *Smooth Streaming*. 2017. URL: https://www.iis.net/downloads/microsoft/smooth-streaming (visited on May 28, 2017) (cit. on p. 19).

[Mic17g] Microsoft Corporation. *Windows Dev Center. Protected Media Path.* 2017. URL: https://msdn.microsoft.com/en-us/library/windows/desktop/aa376846(v=vs.85).aspx (visited on May 24, 2017).

[Mic17h] Microsoft Corporation. *Xamarin*. 2017. URL: https://www.xamarin.com/microsoft (visited on May 28, 2017) (cit. on pp. 14, 127).

[Mot17] Motion Picture Laboratories, Inc. *MovieLabs*. 2017. URL: http://movielabs.com/ (visited on Apr. 18, 2017) (cit. on pp. 7, 28).

[Mov16] Moving Picture Experts Group. 2016. URL: http://mpeg.chiariglione.org (visited on Feb. 12, 2016) (cit. on p. 33).

[Moz17a] Mozilla Foundation. *Gecko*. 2017. URL: https://developer.mozilla.org/en-US/docs/Mozilla/Gecko (visited on May 28, 2017) (cit. on p. 16).

[Moz17b] Mozilla Foundation. *Mozilla Firefox*. 2017. URL: https://www.mozilla.org/firefox/ (visited on May 25, 2017) (cit. on p. 10).

[Nat] National Center for Supercomputing Applications. *NCSA Mosaic™*. URL: http://www.ncsa.illinois.edu/enabling/mosaic (visited on May 28, 2017) (cit. on p. 14).

[Net17a] Netflix, Inc. *Netflix - Watch TV Shows Online, Watch Movies Online*. 2017. URL: https://www.netflix.com/ (visited on May 22, 2017) (cit. on p. 1).

[Net17b] Netflix, Inc. *Netflix System Requirements for HTML5 Player and Silverlight*. 2017. URL: https://help.netflix.com/en/node/23742 (visited on May 24, 2017).

[O. 14] O. Friedrich. *Our Strategy for STB-less IPTV*. 2014. URL: http://iptvseminar.com/documents/2014/11/oliver-friedrich-deutsche-telekom-stb-less-standard.pdf (visited on July 25, 2015).

[OP-16] OP-TEE Repository on GitHub. *OP-TEE build.git*. 2016. URL: https://github.com/OP-TEE/build (visited on June 10, 2017) (cit. on p. 95).

[Opea] Open BSD. *Clock Gettime*. URL: http://man.openbsd.org/OpenBSD-6.0/clock_gettime.2 (visited on June 14, 2017) (cit. on p. 112).

[Opeb] OpenStack. URL: https://www.openstack.org/ (visited on June 11, 2017) (cit. on p. 115).

[Ope16] Open IPTV Forum (OIPF). 2016. URL: http://www.oipf.tv (visited on Feb. 8, 2016) (cit. on pp. 5, 83, 164).

[P. 13] P. Wells. *The Greatest Inflection Point in Technology History*. 2013. URL: http://www.cisco.com/c/dam/global/pt_pt/assets/cisco-connect-2013/pdf/16_outubro_peter_wells.pdf (visited on Apr. 13, 2017) (cit. on p. 14).

[Qua17] Qualcomm. *Secure Foundation*. 2017. URL: https://www.qualcomm.com/products/features/security#secure-foundation (visited on May 29, 2017) (cit. on p. 31).

[S. 07a] S. Di Paola. *Finding Vulnerabilities in Flash Applications*. 2007. URL: http://slidegur.com/doc/5738463/presentation (visited on Oct. 18, 2016) (cit. on p. 19).

[S. 07b] S. Di Paola. *Testing Flash Applications*. 2007. URL: http://slidegur.com/doc/5715/testing-flash-applications (visited on Oct. 18, 2016) (cit. on p. 19).

[S. 08] S. Chatterji. *Flash Security And Advanced CSRF*. 2008. URL: https://www.owasp.org/images/2/2d/FLASH_Security.pdf (visited on Oct. 18, 2016) (cit. on p. 19).

[S. 10] S. Jobs. *Thoughts on Flash*. 2010. URL: http://www.apple.com/hotnews/thoughts-on-flash/ (visited on Oct. 24, 2016) (cit. on p. 20).

[STM17] STMicroelectronics. 2017. URL: http://www.st.com/ (visited on May 29, 2017) (cit. on p. 31).

[Sam16a] Samsung. *Real-time Kernel Protection (RKP)*. Feb. 2016. URL: https://www.samsungknox.com/en/blog/57b23cb977bf294a176adea7 (visited on May 29, 2017) (cit. on p. 31).

[Sam16b] SamyGO. *SamyGO Wiki: MICOM commands*. 2016. URL: http://wiki.samygo.tv/index.php5/Watchdogs_and_MICOM_on_A_series%28SH4%29#Other_Micom_commands (visited on Feb. 12, 2016) (cit. on p. 44).

[Sch16] Scherg, R. *DVB Snoop*. 2016. URL: http://dvbsnoop.sourceforge.net/ (visited on Feb. 16, 2016) (cit. on p. 95).

[Sie17a] Sierraware. 2017. URL: https://www.sierraware.com/ (visited on May 29, 2017) (cit. on p. 31).

[Sie17b] Sierraware. *SierraTEE for ARM® TrustZone®*. 2017. URL: https://www.sierraware.com/open-source-ARM-TrustZone.html (visited on May 29, 2017) (cit. on p. 31).

[Ski17] Skia Inc. *Skia Graphics Engine*. 2017. URL: https://skia.org// (visited on May 28, 2017) (cit. on p. 17).

[Sof17] Software Systems Laboratory. Columbia University Department of Computer Science. *KVM/ARM. An Open-Source ARM Virtualization System*. 2017. URL: https://systems.cs.columbia.edu/projects/kvm-arm/ (visited on May 29, 2017).

[Spo17] Spotify AB. 2017. URL: https://www.spotify.com/ (visited on May 28, 2017) (cit. on p. 26).

[TVe17] TVersity. 2017. URL: http://tversity.com/ (visited on June 10, 2017) (cit. on pp. 69, 111).

[The17a] The jQuery Foundation. *jQuery*. 2017. URL: https://jquery.com/ (visited on May 28, 2017) (cit. on p. 15).

[The17b] The Linux Foundation. *Linux*. 2017. URL: https://www.linux.com/ (visited on May 28, 2017) (cit. on p. 155).

[Tru17a] Trusted Computing Group. 2017. URL: http://www.trustedcomputinggroup.org/ (visited on May 29, 2017) (cit. on p. 30).

[Tru17b] Trusted Computing Group. *Trusted Platform Module (TPM) Specifications*. 2017. URL: http://www.trustedcomputinggroup.org/resources/tpm_library_specification (visited on May 29, 2017) (cit. on p. 30).

[Tru17c] Trustonic. 2017. URL: https://www.trustonic.com/ (visited on May 29, 2017) (cit. on p. 31).

[Twi17] Twitter Inc. *Twitter. It's What's Happening*. 2017. URL: https://www.twitter.com/ (visited on May 22, 2017) (cit. on p. 1).

[Uni16] Universal Plug and Play. *Open Connectivity Foundation (includes all the activities formerly sponsored by UPnP Forum)*. 2016. URL: https://openconnectivity.org (visited on Aug. 16, 2016) (cit. on p. 38).

[V. 16] V. Hosakote. *Multicast in OpenStack*. Sept. 2016. URL: https://www.slideshare.net/Vikram_Hosakote/multicast-in-openstack (visited on June 11, 2017) (cit. on p. 115).

[W3Ca] W3C. *Web App Manifest*. URL: http://www.w3.org/TR/appmanifest/ (visited on June 13, 2017) (cit. on p. 127).

[W3Cb] W3C. *Web Media API Community Group*. URL: https://www.w3.org/community/webmediaapi/ (visited on June 13, 2017) (cit. on p. 128).

[W3C17a] W3C. *Call for MSE V2 and EME V2 Use Cases and Requirements*. 2017. URL: https://lists.w3.org/Archives/Public/public-html-media/2016Jul/0017.html (visited on Jan. 8, 2017) (cit. on p. 109).

[W3C17b] W3C. *Cloud Browser TF*. 2017. URL: https://www.w3.org/2011/webtv/wiki/main_Page/Cloud_Browser_TF (visited on Jan. 9, 2017) (cit. on p. 109).

[Wor16] World Wide Web Consortium (W3C). 2016. URL: http://www.w3.org (visited on Feb. 8, 2016) (cit. on pp. 5, 14, 164).

[dev16] developer.mozilla.org. 2016. URL: https://developer.mozilla.org/en-US/docs/Web/HTML/Supported_media_formats (visited on Oct. 21, 2016) (cit. on p. 19).

[fuk07] fukami Chaos Communication Camp. *Testing And Exploiting Flash Applications*. 2007. URL: https://events.ccc.de/camp/2007/Fahrplan/attachments/1320-FlashSec.pdf (visited on Oct. 18, 2016) (cit. on p. 19).

Appendices

Other Author's Publications

A

The following papers are not part of this thesis but may serve as an additional contribution to and extension of the work presented in this thesis:

P6 A. Mikityuk, J. P. Seifert, and O. Friedrich. „Paradigm Shift in IPTV Service Generation: Comparison between Locally- and Cloud-rendered IPTV UI". In: *2014 IEEE 11th Consumer Communications and Networking Conference (CCNC)*. 2014, pp. 205–212

P7 A. Mikityuk and O. Friedrich. „Securing Virtual Service Generation on the Network: Adapting Digital Rights Management to Cloud-delivered Media". In: *2014 ASE BIGDATA/SOCIALCOM/CYBERSECURITY Conference*. 2014

P8 A. Mikityuk, B. Zachey, and O. Friedrich. „Digital Rights Management and its Evolution in the Context of IPTV Platforms in the Web Domain". In: *2014 IEEE/CIC International Conference on Communications in China (ICCC)*. 2014, pp. 193–198

P9 A. Mikityuk, O. Friedrich, R. Skupin, Y. Sanchéz, and T. Schierl. „Compositing without Transcoding for H.265/HEVC in Cloud IPTV and VoD Services". In: *2015 IEEE 5th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*. 2015, pp. 176–180

Paper 6 identifies the fact that routing UI delivery out of the Cloud with regard to RSE causes a significant impact on the network infrastructure. This might have some implications for system response and can therefore result in a lower Quality of Experience overall. This paper describes this approach and its current implementation, while identifying and addressing network challenges as well.

Furthermore, this paper provides a comparison between the IPTV service executed locally on an end-user device - locally-rendered IPTV UI - and that delivered to an end-user after it has been executed within the Cloud - Cloud-rendered IPTV UI - and shows the feasibility of the presented RSE approach empirically.

Paper 7 introduces a novel content-protection security framework within the RSE concept. The concept poses new challenges for the classical IPTV security mechanism. This work presents a novel security framework for RSE, analyzing potential system vulnerabilities and security threats.

Paper 8 introduces a novel mechanism for content protection in RSE-based IPTV: the architecture presented here integrates Digital Rights Management into the web-based landscape. The platforms for the UI execution studied here are browser-based execution approaches, where the execution environment is provided by the browser. The browser resides on the client-side in the context of browser-based UI execution, whereas, in a fully Cloud-based execution, the browser is shifted into the Cloud.

The client-side implementation for these platforms was performed on the same Set-Top Box hardware, in order to demonstrate the response time difference for both approaches. The response time considered in this paper refers to the UI execution and video encryption for unicast and multicast.

This paper also introduces a breakdown of response time by introducing a method for load-time measurements in the context of RSE. Moreover, the paper presents the evolution of Digital Rights Management systems from the currently existing proprietary approaches to web-based EME.

Finally, Paper 9 brings together novel RSE architectures for media services with cutting-edge, compressed-domain video processing techniques. In this context, this paper presents an alternative to costly and resource-hungry server-side video transcoding, that could be used e.g. for the insertion of advertisement overlays or user interfaces. The concept presented here uses the newly emerged H.265/HEVC standard.

State-of-the-Art

B

B.1 Media Service Execution Environments: OS-based Approach

The OS-based approach might also be referred to as hybrid approach, as both the OS and corresponding Web browser are parts of the MSEE. The most relevant and deployed OS for media services is the Android OS[1]. Android OS is an OS based on the Linux [The17b] kernel and designed for mobile touchscreen devices.

It is supported on various processor architectures and runs on the majority of mobile devices. Google releases the Android's code under an open source license. However, Android devices are a combination of open source and proprietary software required to access Google services.

First released on the market on an Android device in 2008, the Android OS has since had multiple releases. The current Android version is 7.0, and versions below 4.0 are almost completely gone from the market. Still, Android OS versioning remains an issue for Android application developers as the versions between 4.0 and 7.0 are still actively used [Goo17k]. For WMS services, Google has also released its Android TV system that has a dedicated MUI.

The Android OS is composed of Linux kernel, libraries with Android Runtime (ART) and the Application Framework. Linux kernel enables the hardware abstraction and consists of multiple device drivers for video, audio, display, etc.

On top of the kernel, system libraries like Media Framework, Android's C library, OpenGL [Khr17], etc. and ART translate the bytecode of applications into native device instructions. This layer implements APIs used by the Application Framework to run the applications. The applications can be deployed while making use of these APIs by using the Android Native Development Kit (NDK) [And17a] to develop applications in C and C++ or the Android SDK included in Android Studio Integrated Development Environment (IDE) [And17b] to develop in Java.

B.1.1 AndroidTV

In 2014, Google released the Android TV - an OS for Smart TVs and digital media players based on Android OS. This brings the Android ecosystem to TVs that, among other things, enable users to access Google Play Store and download a variety of already available Android applications. The applications also include leading WMS services like Netflix, Amazon, etc. Google implements the voice search that enables quick navigation through the MUI. Android TV is currently deployed on Sony, Sharp and Philips Smart TVs.

[1] Android Inc. developed this system, acquired in 2005 by Google and released in 2007.

B.2 Streaming

B.2.1 Web Streaming

B.2.1.1. HTTP Progressive Download

Progressive download is the most straightforward media distribution technology. It is quite literally a conventional media file transfer between a server and a client, however progressive downloads are perceived by end-users as media streaming. A full media file is downloaded by the client over the HTTP protocol, where the client requests the media file with XHR using the HTTP GET method.

The content in progressive download is distributed by a standard Web server, which simplifies the infrastructure set up and does not require dedicated licenses. Standard Web servers are specifically designed to transfer the data over the Web, serving large amounts of clients with a very efficient usage of available network bandwidth.

The playback of the content would not start immediately, as it depends on the position set for the media file header. If the header is positioned at the beginning of the file, the client would start to playback the content right after the metadata included in the header has been received. Otherwise, the client would wait until the whole file downloads when the header has been placed at the end of the file.

However, even in the first case when the header is set at the beginning of the file, the client player must wait until the first portion of media data becomes available. This in turn depends on the media file encoder parameters, which are set by media content distributors. Furthermore, it is not possible for end-users to jump forward while watching the media, as the data must be downloaded sequentially.

Following the format of a conventional file transfer, the media file downloaded from the server is saved temporarily on a user's hard drive. This poses content security risks for media content distributors, as the complete media file is saved on the end-user's computer.

Therefore, media content can be easily redistributed by end-users at a large scale. Finally, as the files are all downloaded even though end-users might not watch all the content, a significant portion of the used bandwidth might therefore be wasted.

B.2.1.2. Real-time Streaming Protocols

Real-time streaming protocols enable real content streaming, where the content file is divided into smaller segments. These include such protocols as Real Time Transport Protocol (RTP), Real Time Streaming Protocol (RTSP), Real Time Messaging Protocol (RTMP).

These are stateful protocols that maintain server-client connections until a client is disconnected. The segments are delivered from a server to a client over a persistent socket connection maintained until all data has been transferred to the client.

On the other hand, the permanent socket connection required between the server and the client does not allow system scaling for live events. As servers can have only a restricted

amount of such socket connections, real-time streaming requires more streaming servers for the same amount of end-clients as a progressive download or adaptive HTTP streaming.

This connection is mostly UDP-based, which makes a big difference with regard to TCP-based HTTP delivery. UDP is a very lightweight protocol that does not require any data re-transmission for lost packets. This is suitable for video delivery that does not pose harsh requirements for quality and can be compensated with additional technologies, if required.

Segment delivery enables multiple trick play functions. e.g. forward, fast-forward, etc., for end-users, giving them full control over video playback. With regard to content security issues, the real-time streaming provides a greater level of security than progressive download. The small segments of data are delivered to the client upon request, and played back immediately without storing the data on a hard drive.

However, real-time streaming protocols are served by specialized streaming servers, e.g. Wowza. These often require dedicated licensing, more complex set ups than the standard Web servers, and non-common connection ports that must be configured.

B.2.2 Streaming Protocols Summary

Table B.1 summarizes the overview of streaming technologies presented earlier. It also compares these with regard to the following 4 groups of characteristics: general description of technologies, their efficiency, conditions for functioning and security of content provided.

B.3 Protecting Web Streaming

B.3.1 Conditional Access

Conditional Access [Cut97] was introduced by DVB as standard for Digital TV [Dig16b] and focuses on the definition of access rights. The access rights are used to decide whether access to the media content can be granted to a customer or not.

The securing algorithm used in DVB CA for Digital TV is specified by ETSI and is known as Common Scrambling Algorithm (DVB-CSA). DVB-CSA is used for securing the MPEG-2 TS, examined in Chapter 2.3.3.1 in detail. CA systems were introduced for broadcasting systems and therefore designed for one-way communication from a broadcaster to a customer.

DVB Common Scrambling Algorithm

The actual algorithm for scrambling and descrambling in DVB-CSA was not publicly disclosed. Except for a few details [ETS96] [Kim+97] [Kuh+98], manufacturers had access to the DVB-CSA specification [DVB96] only under non-disclosure agreements. DVB-CSA was kept secret until the FreeDec software implementation of CSA appeared on the Web in 2002 [Die12].

The DVB-CSA algorithm was reverse-engineered and researchers started analyzing the algorithm for potential security vulnerabilities [Wir05] [LG07]. The first version of CSA was a combination of a 64-bit block and stream cipher [ETS96].

The more robust version of the CSA algorithm is version 3 and is known as DVB-CSA3 [ETS11]. DVB-CSA3 is based on Advanced Encryption Standard (AES) with the 128-bit

Tab. B.1.: Comparison of Streaming Technologies

Characteristic	HTTP progressive download	Real-time media streaming protocols	Adaptive HTTP streaming protocols
General description			
Technology summary	Simple download of an entire media file, playback at one single data rate	Segment-based streaming of media file segments at one single data rate	Segment-based streaming of media file segments at multiple data rates
Perceived by end-users as	Streaming	Streaming	Streaming with different bitrates
Transport protocol	TCP	Mostly UDP with TCP rollover on the network where UDP is not supported	TCP
Streaming server	Standard HTTP Web Server	Dedicated Streaming Media Server	Standard HTTP Web Server
Efficiency			
Server-client connection type	Stateless, one-time download of an entire media file	Stateful, persistent connection is maintained until a client is disconnected	Stateless, one-time transactions of media segments
Client request processing	Efficient through one-time downloads	Non-efficient through persistent connections with clients	Efficient through one-time transactions
Bandwidth usage	Bandwidth might be wasted as not all downloaded media files may be watched	Bandwidth saving through segment-based delivery	Bandwidth saving through segment-based delivery and bandwidth optimization through CDN caching for reusage by multiple clients
Conditions			
Preparation of content	Not required	Streaming media server is responsible for handling segments	Content must be encoded at multiple bit rates
Encoding bitrates	Content can be encoded at higher data rates R_{encode} than available client bandwidth B_{clietn}. I.e., if Equation B.1 is true, the download time increases but content still can be played back. $R_{encode} > B_{client}$ (B.1)	Content must be encoded at data rates R_{encode} less or equal to available client bandwidth B_{clietn}, as stated in Equation B.2. Otherwise, playback would be stopped. $R_{encode} \leqslant B_{client}$ (B.2)	Content is encoded at multiple data rates. Dependent on the available client bandwidth B_{clietn}, client requests the segments with data rates R_{encode} that satisfy the Equation B.2. Continuous playback is ensured, as multiple R_{encode} are available.
Bad network conditions robustness	Download time of media files increases reasonably	Playback is interrupted, unless media content provider enables a lower quality option to be chosen by end-users	No playback interruption, content is immediately available at lower quality
Real-time retrievement	Not required, if content has been already downloaded then connection does not play a role	Is required	Is required
Content security			
Securing of content	Media files are saved on a hard drive of an end-user's machine	Improved security, as media files are not stored locally	Improved security, as media files are not stored locally

key length combined with DVB confidential eXtended emulation Resistant Cipher (XRC). However, slow market adaptation of the DVB-CSA3 due to a legacy hardware basis has not yet enabled the complete replacement of DVB-CSA.

DVB Conditional Access

The main components of CA systems currently implemented in broadcast delivery platforms include:

- **CA System:** CA scrambling system that generates keys and makes them available for the Scrambler;
- **Secure Client:** A client-side software or hardware mechanism that enables the de-scrambling of scrambled media following the corresponding scrambling scheme;
- **Scrambler:** Encodes media, then scrambles it with keys provided by the CA System and prepares the content for distribution.

On the broadcaster premises (i):

The data payload that is transported in MPEG-2 TS is scrambled by a key or a so-called Control World (CW) [Ben14] [Man]. The process of scrambling changes some of the bit values, according to a certain algorithm, to enable security.

The CWs cannot be transmitted over the network as plain text and therefore are encrypted by the CA system (CAS) to the so-called Entitlement Control Messages (ECMs). The decision information on the access conditions required by a decision logic that resides on the client is included in Entitlement Management Messages (EMMs). This information contains data that specifies whether access to a media file must be granted to a user or not.

The CWs are sent to CAS from the head-end over the so-called DVB SimulCrypt interface [ETS08b]. The same interface is responsible for sending ECMs and EMMs back to the head-end, after being generated in CAS. In the head-end, the ECMs and EMMs are multiplexed with the scrambled MPEG-2 TS packets that contain audio, video and text data.

On the user premises (i):

After the scrambled content has been transmitted over the network in the MPEG-2 TS stream, the stream is demultiplexed. The client middleware recognizes that the stream is encrypted by getting program-specific information out of the stream during demultiplexing.

The client middleware parses the Program Map Table and when it contains an identifier of an ECM, the stream is encrypted [WW05]. The middleware sends ECM identifiers, ECMs and EMMs to the Secure Client. The Secure Client is either an obfuscated software or hardware implementation of a certain CAS decryption algorithm.

The hardware implementation is usually a secure processor on a smart card that is located in the Conditional Access Module (CAM). The CAM needs a dedicated Personal Computer Memory International Association (PCMCIA) slot to be connected to a client.

The client middleware sends the data to the CAM through the Common Interface (CI) [ETS99]. The latest version of the CI is CIPlus version 1.4 [CI 16], which will appear on the market in 2017. The Secure Client, using the received ECM identifier, applies the correct decryption key. In pure broadcast environments with one-way communication, keys are sent

to clients at certain time intervals through Digital Storage Media Command and Control (DSM-CC), which is studied in detail in Chapter 2.3.3.1.

Regarding the bi-directional CAS systems, if there is no key in secure storage, the client will make a request to key servers. The bi-directional CAS systems evolved with the advent of such technologies as Internet Protocol Television (IPTV) [Xia+07] and Hybrid Broadcast Broadband TV (HbbTV) [Hyb16].

After the ECM is decrypted, the obtained CW is sent to the Descrambler, which will descramble the media. The descrambled media is then forwarded to the graphics pipeline, including the video decoder and graphics engine.

B.3.1.1. CA Key Decryption Hierarchy

A so-called key ladder is a content key decryption hierarchy that is based on a master key. The master key is a unique hardware identifier that is stored either on a chip of the device itself or a CA smart card chip.

The key hierarchy might consist of multiple levels. ETSI provides the specification for a three-level key ladder (K-LAD) [ETS10a].

For example, in this concept, the content keys are encrypted with the above-mentioned master key. The content keys, in turn, can decrypt the next level key, e.g. channel keys. The channel keys further decrypt channel group keys, etc. The amount of such K-LAD hierarchy decryption levels might depend on CA provider, implementation, etc.

B.3.2 Digital Rights Management

Digital Rights Management is a content protection technology required by content providers for content streaming between the media streamer and the end-user. With DRM the owner of the content has full control over access to the content, beginning at its creation and extending up to the point of use of the content.

To provide users with more rights for Web content usage that they have purchased from media distributors, DRM has introduced some licenses [Liu+03]. Analogue to EMMs in CA, as described in Chapter B.3.1 where user access rights are defined, DRM systems define licenses which stipulate what rights exactly users have for certain pieces of content.

The licenses are data files that define usage rights, including age restrictions for content watching, allowed devices, maximum number of titles that can be saved on a device, etc. For example, contents could be either purchased or borrowed; if purchased then they could be used either on one or multiple devices; if borrowed then either for one day or the whole week.

In summary, usage rights depend on the content and application owner, application type, e.g. video or audio, the content type, e.g. premium video or free-to-air channels, and, last but not least, the DRM provider and its technical solution. Herewith, DRM enables content rights, their enforcement and corresponding payment systems.

The main components of DRM systems currently implemented in WMS platforms include:

- **DRM System:** A DRM encryption system that generates Content Encryption Keys (CEKs) and makes them available for the Encryptor;
- **Secure Client:** A client-side crypto function - software or hardware mechanism that enables decryption of encrypted media following the according encryption scheme;
- **License Server:** Interacts with a Secure Client to provide keys and licenses to decrypt media;
- **Encryptor:** Encodes media, then encrypts it with CEKs provided by the DRM System and prepares content for distribution.

Encryptors and DRM Systems are usually located at the media provider head-end. License servers, where clients request for keys and licenses, are physically located closer to the clients. Finally, the Secure Client is part of an end-device.

The following description will present the fundamental operation logic of any DRM system used to protect Web content. As mentioned earlier in Chapter 2.2.2.3, DRM systems are proprietary technologies and every manufacturer has its own technology.

There are multiple proprietary DRM schemes used in the field, which are either device or smart/sim-card based and implement different cryptographic mechanisms [Liu+03]. However, all of these systems have a similar operation logic when their high-level structures are described.

On the media distributor premises (i):

Web media distributors receive contents from content providers and are responsible for their preparation. In some cases, content providers and media distributors are the same entity. Therefore, media distributors own Encryptors that prepare media content. This component also might be referred to as a Packager, which is responsible for complete content preparation.

However, to highlight its main function from the content securing perspective - encryption - here it will be referred to as Encryptor. Plaintext content is fed into the Encryptor. The Encryptor encodes the content according to the required output codec. Here the content might also be transcoded if the input codec does not satisfy output requirements.

After this operation, the DRM System provides the CEKs to the Encryptor. Every CEK is identified by a unique identifier known as Key ID or KID. The DRM System also provides the so-called DRM system IDs that point to the license servers, where the CEKs corresponding to every piece of content are stored.

The content is encrypted with the CEK provided by the DRM System and prepared for distribution, e.g. stored as one media file, segmented, packetized, etc., depending on the streaming technology. In this step, the metadata, such as content IDs, are also created. The DRM system IDs and KIDs are also embedded into the content and will be shared with clients. Finally, when its ready for distribution, the content is handled by streaming servers, described in detail in Chapter 2.2.1 for different streaming technologies.

On the user premises (i):

Clients receive the media content. Analogue to CA, the graphics library that is part of a client's middleware parses the content. Based on the content metadata, the client recognizes that a license is required for playback. The invoked Secure Client requests the license at the License Server of the DRM System, according to the DRM System ID and KID.

The license request also includes content IDs, user's authentication information and user's device information, e.g. device public key. The licenses also might be locally pre-stored within the Secure Client. In this case, the communication with license servers does not take place. The Secure Client includes the initialization mechanisms that generate private and public keys for user's device.

On the media distributor premises (ii):

The License Server receives the license request and validates all the parameters sent by the client. It authorizes the user and its device for a certain usage and certain content IDs. The server encrypts the CEK with a public key from the user's device. It creates the license in which usage rights are defined and which contains the encrypted CEK.

On the user premises (ii):

The Secure Client, having received the encrypted CEK, decrypts it with the device's private key or a so-called master key. The Secure Client also has a DRM logic that, depending on the usage rights, defines and enforces the decryption rules. After the CEK is decrypted, it is applied to the encrypted content.

B.3.2.1. DRM Key Decryption Hierarchy

Analogue to the Key Ladder technique in CA, a so-called Key Rotation is used in DRM to distribute live content. The master key that is stored on a chipset of user's device hardware is used to decrypt further keys. These keys cannot be directly applied for content decryption. The master key, or the CEK, decrypts the so-called rotation keys. The rotation keys are embedded into the content and are rotated at regular intervals.

B.3.2.2. Common Encryption

The Common Encryption INSO/IEC 23001-7 specification (CENC) [ISO16] standardizes encryption and key mapping techniques. These can be utilized to decrypt protected MPEG-DASH streams, explained in detail in Chapter 2.2.1.1, using different DRM systems.

The standard specifies encryption algorithms and a common format required for the decryption of metadata. Metadata include DRM system IDs, key server addresses (URLs), keys and licenses and are DRM-specific. All other details, e.g. key acquisition or storage techniques, are left up to DRM systems. CENC specifies encryption for file formats based on INSO/IEC 14496-12, INSO Base Media File Format.

The above-mentioned metadata including DRM rights, licenses with keys and key acquisition information, are stored in media within the so-called Protection System Specific Header (PSSH) boxes [Dor+14]. The information stored in media data is used by different DRM

systems on a client device, which, in turn, correspond to different DRM system IDs. Due to this information, a client is notified that the content is encrypted.

CENC specifies an encryption algorithm using AES-128 in both AES Counter Mode (CTR) and Cipher Block Chaining (CBC). The third edition of CENC reduces the computational power of clients through the introduction of pattern encryption techniques CENS (AES-CTR mode pattern encryption) and CBCS (AES-CBC mode pattern encryption). The sample encryption encrypts only a fraction of the data contained in each video sample.

B.3.2.3. DRM Secure Client

Three ways of creating clients for media streaming exist: (1) native, (2) Web and (3) hybrid applications. The tools and programming languages for creating these three kinds of applications are, respectively, (1) native platforms technologies, (2) Web technologies and (2) a mixture of both. Therefore, the two main groups of technologies comprise native and Web tools.

Either way, Secure Clients for DRM systems might be implemented in software, hardware-assisted software or hardware. However, for Web applications, DRM-secure clients are implemented through a Browser plugin. For native applications, the corresponding clients are native clients for embedded platforms.

With regard to Web applications, until recently the plugin technology was a widespread solution for desktop browsers, where the Secure Client of a DRM system was implemented within a Browser plugin on a client. This therefore has not posed any restrictions on multiple DRM environments.

These plugins were working with the majority of browsers over Netscape Plugin API (NPAPI). Through its almost ubiquitous support, NPAPI was perceived as a cross-platform interface for DRM plugins. Analogue to NPAPI, ActiveX was provided as a plugin interface by Microsoft Internet Explorer. For example, the Microsoft PlayReady DRM component inside the Silverlight plugin and the Adobe Access DRM component inside Flash Player were used for content decryption.

However, in 2013 browser manufacturers started the process of deprecation of NPAPI [the] [tec]. In 2016, support for the interface has been discontinued by almost all Web browsers. To provide plugin functionality, a new, more secure interface Pepper Plugin API (PPAPI) was introduced by Google.

However, plugin implementations of DRM secure clients through PPAPI are not certified by manufacturers on Google, Apple and Microsoft platforms. To address the integration of crypto libraries with browsers, a new, alternative interface - Encrypted Media Extensions - was presented by HTML5. It became the most adopted interface for DRM secure clients on the Web in 2016.

B.4 Hybrid Broadcast Broadband TV

B.4.1 Hybrid TV and Related Standards

The HbbTV standard is based on the existing standards of the Open IPTV Forum (OIPF) [Ope16], Consumer Electronics Association (CEA) [Con16a], Digital Video Broadcasting (DVB) [Dig16b], and World Wide Web Consortium (W3C) [Wor16]. HbbTV utilizes the common Web technology CE-HTML (Consumer Electronics Hypertext Markup Language), which in turn supports JavaScript, Cascading Style Sheets (CSS) and the Document Object Model (DOM).

Version 1.1.1 of the HbbTV specification referenced parts of the standards and specifications available at that time. The specification of the application signaling and its transport over broadcast or broadband is provided by DVB TS 102 809 [ETS10c].

The OIPF Declarative Application Environment (DAE) [Ope16] specifies the JavaScript APIs for TV applications and defines how to embed linear A/V content into these applications. CEA-2014 [CEA11] defines the application language (Extensible HyperText Markup Language (XHTML), CSS, JavaScript, AJAX), DOM event-handling, and embedding of non-linear A/V content into an application. XHTML, DOM, and CSS are specified in W3C.

HbbTV 1.5 [ETS12] adds additional features by referencing MPEG Dynamic Adaptive Streaming over HTTP (DASH) [ISO12a] and MPEG Common Encryption (CENC) [ISO12b]. With MPEG DASH, the HbbTV alliance introduces support for HTTP adaptive streaming.

The DASH-based, non-linear content can be protected with multiple DRM schemes based on the Common Encryption mechanism specified by MPEG CENC. This version also includes the DVB-SI section of the OIPF DAE specification, which provides better access to the broadcast scheduling information.

A high-level overview of the relationship between the standards is presented in Figure B.1. The HbbTV 2.0 specification released in 2015, introduced the support of companion devices and better synchronization between broadcast and broadband signal.

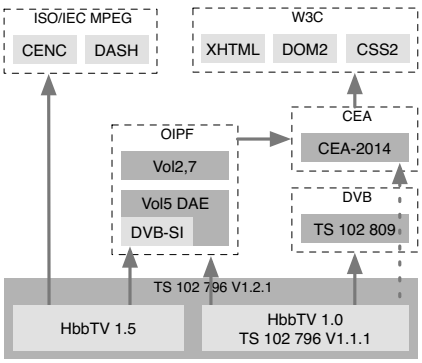


Fig. B.1.: Relationship between standards used in HbbTV (adapted from [ETS12])

B.4.2 Broadcast-Independent Applications

The broadcast-independent applications are not related to any TV channels. These could be any TV-specific application, e.g. Electronic Program Guides (EPGs) or streaming services like YouTube, Netflix, Amazon Lovefilm, etc. Broadcast-independent applications do not require any signaling, as specified by the HbbTV standard [ETS15].

The standard also does not define the processing of the broadcast-independent applications. Instead, the details of this are left to implementations made by Consumer Electronics (CE) manufacturers. However, if the applications are explicitly signaled by providers, the standard specifies identification, launching and possible delivery of the applications.

Identification: the application is identified through the application URL. This URL either refers to the HTML application page or to a so-called XML AIT. The XML AIT is the AIT encoded in XML format that is used for broadband networks [ETS10c], [ETS09b]. However, MPEG-2-encoded in-band AIT tables could not be used simultaneously with XML-encoded AITs in one stream.

Launching: the broadcast-independent application could be launched either through already running, broadcast-related applications or Internet TV Portals. Provided by CE manufacturers or HbbTV operators, Internet TV Portals are start pages for applications. The application URL could also be manually entered or added by the user as applications on mobile platforms.

Delivery: the application-associated data is accessed via broadband. The HTML application page or XML AIT could be delivered over HTTP or HTTPS. The XML AIT could also be provided to the HbbTV Terminal by a HbbTV Companion Screen.

B.4.3 HbbTV Device Fragmentation

B.4.3.1. DVB Transmission

The DVB project specifies different physical channels for broadcast delivery of baseband signals: satellite, cable and terrestrial. The terrestrial transmission for DVB is specified in EN 300 744 [ETS09a] and is known as DVB-T.

DVB-T transmission is based on the Orthogonal Frequency-Division Multiplexing (OFDM) technology. OFDM is a modulation method that utilizes multiple carrier frequencies for data encoding. DVB-T supports three modulation schemes to modulate the carriers for OFDM: QPSK, 16-QAM and 64-QAM. The bandwidth of DVB-T channels used in most countries is $7\,\mathrm{MHz}$ for the VHF band and $8\,\mathrm{MHz}$ for UHF.

DVB-S [ETS97] uses Quaternary Phase Shift Keying (QPSK). DVB-C [ETS98] transmission uses Quadrature Amplitude Modulation (QAM) with 16, 32, or 64-QAM constellations, with possible future extensions for 128-QAM and 256-QAM.

B.4.3.2. HbbTV-enabled Device Stack

A stack from a current HbbTV-enabled device is presented in Figure 2.6. The device must be equipped with the Broadcast interface to be able to receive broadcast streams. The Broadcast

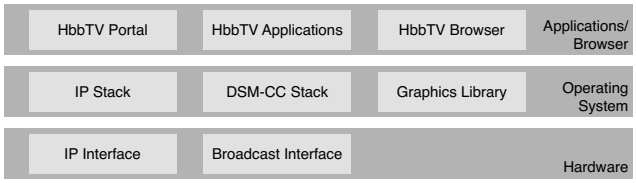


Fig. B.2.: HbbTV-enabled Device: Current Stack

interface must be able to receive the DVB transmission standard specified by the Broadcaster, as mentioned in Annex B.4.3.1.

The broadcast data can also be obtained over the IP interface. However, this can only take place when the IPTV provider generates special IPTV streams that are compliant with the HbbTV standard and provide the DVB data.

To process these data, the DSM-CC logic must be implemented - DSM-CC stack - on the end-device. The DSM-CC was described in detail in Chapter 2.3.3.1. To execute HbbTV applications within the HbbTV Portal, the browser on the end-device must be HbbTV compliant. This means that the browser must contain a so-called HbbTV profile, i.e. HbbTV libraries. The HbbTV libraries were presented in detail in Chapter 2.3.3.2.

The landscape of HbbTV device fragmentation is presented in Table B.2. The fragmentation is analyzed with regard to the STB market in IPTV and HbbTV domains and the mobile market. In this table, the devices are examined using the following criteria:

- *DVB support:* the device is capable of receiving and processing the DVB data
- *HbbTV execution:* the browser on the device is HbbTV-enabled
- *General browser support:* the device is able to execute rich multimedia applications and latest HTML5 technologies within the browser
- *Streaming:* the device either supports streaming over the RealTime Control Protocol (RTCP) or one of the Adaptive Streaming standards (e.g. DASH, HLS)
- *Encoding:* the device supports either of the H.264 or H.265 video encoding technologies

These criteria introduce the device fragmentation that the HbbTV standard cannot address.

In summary, the following device groups can be identified based on these criteria:

1. *Legacy devices and future low-cost devices:* there are many legacy devices on the market which do not support HbbTV and which cannot be upgraded due to hardware limitations. These devices are also not able to execute rich multimedia applications within the browser. This is specifically the case for Hybrid TV and IPTV STBs that were deployed some years ago and cannot be easily exchanged for commercial reasons. This is also the case for low-cost devices.
2. *Current devices:* the HbbTV 2.0 specification introduces a number of new features which are not supported by HbbTV 1.0/1.5. As an upgrade of already deployed HbbTV 1.0/1.5 devices is not realistic due to technical and commercial issues, fragmentation even within HbbTV deployments will be the unavoidable consequence.

Tab. B.2.: Fragmentation of HbbTV Connected Devices

Device category		DVB support	HbbTV execution		General browser support	Streaming		Encoding	
Type	Device		1.X	2.0		RTCP	Adaptive Streaming	H.264	H.265
Legacy devices	Legacy IP STB	no	no	no	no	yes	no	yes	no
	Legacy Hybrid STB	yes	yes	no	no	yes	no	yes	no
Future low-cost devices	Low-cost IP STB	no	no	no	no	n.s.	yes	n.s.	yes
	Low-cost Hybrid STB	yes	yes	yes	no	n.s.	yes	n.s.	yes
Current devices	IP STB	no	no	no	yes	either/or		yes	no
	Hybrid STB	yes	yes	no	yes	either/or		yes	no
Mobile		no	no	no	yes	either/or		either/or	

3. *Mobile devices:* are rapidly gaining in focus for the reception of broadcast or live streaming in a hybrid environment. This device class is not well suited to implement the HbbTV standard designed for CE devices.

B.5 Summary of Related Work

The assessment of the most significant related works is presented for SRSE, SES and SIC services in Tables B.3, B.4 and B.5, respectively. These tables briefly describe the *scope* of each publication and its main application case in *application*.

The assessment of the papers is done with the help of *research questions* that are identified as RWRQ (Related Work Research Questions) and defined as follows:

SRSE Assessment Methodology

- **RWRQ1** *Remote Service Execution (RSE)* studies and analyzes RSE architectures and their feasibility for certain applications;
- **RWRQ2** *Distributed execution of functions* studies RSE architectures that offload not a complete set of functions but rather a smaller function sub-set to a remote host for execution. The options considered in this research question are *peer-to-peer*, *client-to-server* and *server-to-client* offloading. The first one analyzes the function distribution between equally privileged machines or peers. The second one considers the offloading from a client machine to a remote server, and the last one from a remote server to a client machine;
- **RWRQ3** *Distributed execution within a single function* considers more granular RSE architectures, where the shift of the execution is partially accomplished within a

single function. Here, the distribution might be either flexible or permanent. For flexible distribution, there is a logic to deciding in which way the distribution must be performed at a certain point of time, depending on available computational resources, available storage, additionally required appliances, etc. When considering permanent distribution, on the other hand, the distribution scheme remains permanent and the sub-tasks within one function are assigned to the machines participating in distributed execution in a constant manner;

- **RWRQ4** *Security of RSE* analyzes security aspects of RSE architectures. These include possible security implications for system functions due to distributed execution, also when considering the security functions itself, vulnerabilities that could potentially arise with RSE and overall system security;

- **RWRQ5** *Virtual environments threats* examines potential security threats of virtualization technologies. The ones considered in this thesis are the threats of *malicious hypervisors* and *malicious virtual machines*, as these elements are the main building blocks for virtualization. Under the condition that these are vulnerable and could be compromised, partial or even the entire service execution could be jeopardized. Therefore, these kind of potential security threats must be always taken into consideration when applying virtualization technologies;

- **RWRQ6** *Side channel/ Covert channel attacks* focuses on researching new and unexpected side and covert channel attacks that could arise in virtualized environments.

Secure Streaming Services Assessment Methodology

- **RWRQ7** *Interoperability of Content Protection (CP)* analyzes concepts and architectures for interoperable content protection technologies. As current content protection techniques, studied in detail in Chapter 2.2, are device and eco-system dependent, it's important to research approaches for the interchangeability of such technologies.

  This means that multiple content protection techniques could be made available on a certain hardware platform, which would enable more flexible delivery architectures and promote propagation of the content on the open Web.

- **RWRQ8** *Privacy of CP* examines users' privacy aspects of CP technologies. Along with the CP metadata exchanged between a server and a client, the protection of users' online activity is of high importance as users do not want to make this data available publicly. Tracking of online users' activities by non-authorized parties must also be prohibited.

  Therefore, this research question analyzes mechanisms that could prohibit the non-legitimate spread of user data on the Internet. Such mechanisms could involve privacy preserving methods used for data transmission between users and Internet services, access restrictions to specific end-device services, etc.

- **RWRQ9** *Addressing security issues of CP* examines techniques for addressing different vulnerabilities in current CP technologies. As valuable content still remains the number one target for attackers, these vulnerabilities must be addressed either by additional security mechanisms or by completely new security architectures.

- **RWRQ10** *Virtualization of CP* studies virtualization approaches of CP. On one hand, this could be applied by content providers to achieve CP interoperability and interchangeability.

  On the other, it could also be misused by attackers to get a non-authorized access to valuable content, as some CP systems might become vulnerable when virtualization is applied. Two possible application use cases are discussed with regard to this question: virtualization *on end-device* and *on back-end*.

- **RWRQ11** *Security issues of CP virtualization* analyzes CP vulnerabilities when virtualization is applied, as mentioned in **RWRQ10**.

- **RWRQ12** *Multiple CP schemes* examines possible approaches for the co-existence of multiple CP schemes on one physical machine. This research question also addresses CP interoperability.

  However, in contrast to **RWRQ7**, the CP schemes are not interchangeable with each other, but are allowed to run in parallel without any impact on the respective security of each of the schemes. This question analyzes CP schemes' co-existence *on end-device* and *on back-end*.

- **RWRQ13** *CP execution in Trusted Execution Environments (TEE)* focuses on approaches for CP schemes' hardening through its execution in TEE. This enforces advanced security measures and therefore enables higher protection levels of CP schemes. Here, both research areas are taken into consideration: the publications that directly address CP schemes and the general TEE approaches that could be used for CP applications.

  In this research question, two TEE approaches are taken into consideration. First, TEE might be enabled as a *software function with hardware assist* that implies a predominate software-based execution with additional hardware support.

  In other words, these could be security-on-a-chip solutions that provide hardware accelerators for cryptographic operations that are executed in software. These solutions might also provide non-volatile memory to store keys and random number generators.

  In summary, software-based execution with hardware assistance enables additional access to hardware resources and is considered more flexible, but also less secure when compared to purely *hardware functions*. *Hardware function* implies new hardware designs and implementations that provide additional hardware security functions that are currently not state-of-the-art hardware technologies.

- **RWRQ14** *Application execution in TEE* analyzes the feasibility of execution of different applications within TEE. Here, *full* and *partial* execution is considered. The first one examines the entire application code execution within the TEE, whereas the latter one concentrates on a distributed execution, where only the security-sensitive parts of an application code run in the TEE.

Secure Interactive Services (INS) Assessment Methodology

- **RWRQ15** *Remote service execution (RSE) for interactive services*, **RWRQ16** *Distributed execution of functions*, **RWRQ17** *Distributed execution within a single function* and **RWRQ18** *Security of RSE for interactive services* are analog to the **RWRQ1**, **RWRQ2**, **RWRQ3** and **RWRQ4** respectively and applied to the RSE for interactive services;

- **RWRQ19** *third-party openness of INS* analyzes an INS platform openness towards third-party providers, as these platforms tend to be quite closed. The openness might be enabled by design when open Web technologies are used or through clearly identified interfaces;
- **RWRQ20** *Privacy of INS* examines how a platform secures private user data and whether or not this information is accessible by adversaries, analog to **RWRQ8**;
- **RWRQ21** *Security of INS* analyses vulnerabilities and corresponding security exploits that exist in current INS standards and implementations;
- **RWRQ22** *Device security of INS* analyses security threats towards INS platforms and possible vulnerabilities in currently deployed INS end-devices.

Tab. B.3.: Secure Remote Service Execution: Assessment of Related Works

◐ – the research question is partly or briefly discussed in the paper; ◑ – the proof-of-concept implementation to evaluate the research question is presented; ○ – the research question is beyond the scope of the paper, however, the paper is related to the research question; ● – the research question is addressed in the paper; *n/a* (not applicable) – the research question is not within the scope of the paper.

	Zhu et al. [Zhu+11]	**Lee et al. [Lee+12]**	**Chun et al. [Chu+11]**	**Liao et al. [Lia+16]**	**Suksomboon et al. [Suk+15]**	**Chow et al. [Cho+09]**	**Schoo et al. [Sch+11]**	**King et al. [KC06]**	**Rutkowska et al. [RT08]**	**Chen et al. [Che+10]**	**Ristenpart et al. [Ris+09]**	**Mikityuk**
Scope	Light weighting of server-based pre-rendering through a peer-assisted rendering concept	Virtualization of Web applications to enable them on legacy STBs	CloneCloud: Distributed execution of unmodified mobile applications on mobile devices and in the Cloud to enable task offloading	LiveRender: Shifting part of gaming computations to the client for local rendering	Identification of virtualization options for Customer-Premises Equipment (CPE)	Survey paper	Survey paper	Proof-of-concept virtualized rootkit attack	Proof-of-concept virtualized rootkit attack that does not require a reboot of the system	Survey paper	Side channel proof-of-concept attack between two virtual machines on the same physical hardware	SRSE with Secure Streaming and Secure Interactive Contents services
Application	On-line gaming	IPTV STB	Mobile devices	Cloud gaming	STB	Cloud security	Cloud networking security	Virtualized rootkits	Virtualized rootkits	Cloud computing security	Cloud security - side and covert channel attacks	Web content streaming platforms
Research questions												
RWRQ1 Remote service execution (RSE)	●◑	● for Web-application execution	●◑	●◑	●◑	n/a	n/a	n/a	n/a	n/a	n/a	●◑
RWRQ2 Distributed execution of functions												
Peer-to-peer	●◑	○	○	○	○	n/a	n/a	n/a	n/a	n/a	n/a	○

Continued on next page

Tab. B.3.: Secure Remote Service Execution: Assessment of Related Works (continued)

	Zhu et al. [Zhu+11]	**Lee et al. [Lee+12]**	**Chun et al. [Chu+11]**	**Liao et al. [Lia+16]**	**Suksomboon et al. [Suk+15]**	**Chow et al. [Cho+09]**	**Schoo et al. [Sch+11]**	**King et al. [KC06]**	**Rutkowska et al. [RT08]**	**Chen et al. [Che+10]**	**Ristenpart et al. [Ris+09]**	**Mikityuk**
Client-to-server	○	● local video rendering	●◑	○	●◑	n/a	n/a	n/a	n/a	n/a	n/a	●◑
Server-to-client	○	○	○	●◑	○	n/a	n/a	n/a	n/a	n/a	n/a	○
RWRQ3 Distributed execution within a single function	●◑ rendering	○	●	●◑	○	n/a	n/a	n/a	n/a	n/a	n/a	●
RWRQ4 Security of RSE	○	○	○	○	◐	◐	n/a	n/a	n/a	n/a	n/a	●◑
RWRQ5 Virtual environments threats												
Malicious hypervisors	n/a	n/a	n/a	n/a	n/a	◐	◐	●◑	●◑	◐	○	○
Malicious virtual machines	n/a	n/a	n/a	n/a	n/a	◐	◐	○	○	◐	●◑	○
RWRQ6 Side channel/ Covert channel attacks	n/a	n/a	n/a	n/a	n/a	○	○	○	○	◐	●◑	○

Tab. B.4.: Secure Streaming: Assessment of Related Works

◐ – the research question is partly or briefly discussed in the paper; ◑ – the proof-of-concept implementation to evaluate the research question is presented; ○ – the research question is beyond the scope of the paper, however, the paper is related to the research question; ● – the research question is addressed in the paper; *n/a* (not applicable) – the research question is not within the scope of the paper.

	Ghodke et al. [GF04]	**Lee et al. [Lee+13]**	**Zou et al. [Zou+10]**	**Petrlic [Pet12]**	**Villegas et al. [Vil+12]**	**Aciicmez et al. [Aci+09]**	**Thekkath et al. [The+00]**	**Suh et al. [Suh+03]**	**Santos et al. [San+14]**	**Frenzel et al. [Fre+10]**	**Hussin et al. [Hus+05]**	**Mikityuk**
Scope	DRM system vulnerabilities in virtual environments	DRM-as-a-Service: content protection in the Cloud	Mobile DRM system that utilizes a sim-card on the mobile device for key storage	Improved proxy re-encryption scheme to provide anonymity for users and reduce the possibility of user profile building	Forensic mechanism for detecting the source of the leakage	Framework for trusted DVB STBs with multiple DVB CAS systems on a single system	Execute-only memory (XOM) hardware implementation	Hardware design for secure application execution	A concept of Trusted Language Runtime (TLR) that uses TrustZone for protection of .NET mobile applications	Integration of a microkernel system with the TrustZone platform	Usage of the TrustZone technology to store encrypted keys for ticketing systems	SRSE with Secure Streaming and Secure Interactive Contents services
Application	Compromising DRM	Content streaming	Content protection (CP) on mobile	DRM for software licensing in Cloud-computing scenarios	CAS with bidirectional network connection	DVB STB with support for multi-CAS	Copy and Tamper resistant software	Secure processors	Mobile application security	Full virtualization for embedded devices	Mobile ticketing system	Web content streaming platforms
Research questions												
RWRQ7 Interoperability of content protection (CP)	◐	n/a	●◑	n/a	●	●	n/a	n/a	n/a	n/a	n/a	●◑

Continued on next page

Tab. B.4.: Secure Streaming: Assessment of Related Works (continued)

	Ghodke et al. [GF04]	Lee et al. [Lee+13]	Zou et al. [Zou+10]	Petrlic [Pet12]	Villegas et al. [Vil+12]	Aciicmez et al. [Aci+09]	Thekkath et al. [The+00]	Suh et al. [Suh+03]	Santos et al. [San+14]	Frenzel et al. [Fre+10]	Hussin et al. [Hus+05]	Mikityuk
RWRQ8 Privacy of CP	n/a	n/a	●◑	●◑	n/a	n/a	n/a	n/a	n/a	n/a	n/a	○
RWRQ9 Addressing security issues of CP	●	n/a	●◑	n/a	●	●	n/a	n/a	n/a	n/a	n/a	●◑
RWRQ10 Virtualization of CP												
On end-device	●◑	○	n/a	n/a	●	●	n/a	n/a	n/a	n/a	n/a	○
On back-end	○	●	n/a	n/a	○	○	n/a	n/a	n/a	n/a	n/a	○
RWRQ11 Security issues of CP virtualization	●◑	○	n/a	n/a	○	●	n/a	n/a	n/a	n/a	n/a	○
RWRQ12 Multiple CP schemes												
On end-device	n/a	○	○	n/a	○	●	n/a	n/a	n/a	n/a	n/a	◐
On back-end	n/a	●	○	n/a	○	○	n/a	n/a	n/a	n/a	n/a	○
RWRQ12 CP execution in TEEs												
Software function with hardware assist	n/a	n/a	●◑	n/a	n/a	●	●◑	●◑	●◑	●◑	●	●◑
Hardware function	n/a	n/a	○	n/a	n/a	●	●◑	●◑	○	○	○	○

Continued on next page

Tab. B.4.: Secure Streaming: Assessment of Related Works (continued)

	Ghodke et al. [GF04]	Lee et al. [Lee+13]	Zou et al. [Zou+10]	Petrlic [Pet12]	Villegas et al. [Vil+12]	Aciicmez et al. [Aci+09]	Thekkath et al. [The+00]	Suh et al. [Suh+03]	Santos et al. [San+14]	Frenzel et al. [Fre+10]	Hussin et al. [Hus+05]	Mikityuk
RWRQ13 Application execution in TEE												
Full	n/a	n/a	○	n/a	n/a	●	●◑	●◑ App execution in TEE can be suspended	○	●◑	○	●
Partial	n/a	n/a	●◑	n/a	n/a	○	○	○	●◑	○	●	●◑

Tab. B.5.: Secure Interactive Services: Assessment of Related Works

◐ – the research question is partly or briefly discussed in the paper; ◑ – the proof-of-concept implementation to evaluate the research question is presented; ○ – the research question is beyond the scope of the paper, however, the paper is related to the research question; ● – the research question is addressed in the paper; *n/a* (not applicable) – the research question is not within the scope of the paper.

	Cheng [Che14]	**Howson et al. [How+11]**	**Franke et al. [Fra+14]**	**Ghiglieri et al. [GT14]**	**Michele et al. [MK14]**	**Lee et al. [Lee+13]**	**Mikityuk**
Scope	Cloud-based media processing platform for live broadcasting services	Distributed media processing and delivery model for HbbTV services	New application eco-system with interfaces between platform operators and application developers for virtual STB	Privacy drawbacks within the HbbTV standard	Vulnerabilities in the media-detection functionality of media players in Smart TVs	Multiple exploitable bugs in Smart TVs	SRSE with Secure Streaming and Secure Interactive Contents services
Application	HbbTV, Broadcasting services	3D video delivery and other rich-media services for HbbTV	Cloud APIs for third-party applications for virtual STB	HbbTV	Smart TVs as HbbTV terminals	Smart TVs as HbbTV terminals	Web content streaming platforms
Research questions							
RWRQ14 Remote service execution (RSE) for Interactive Services (INS)	●◑	●◑	●◑	n/a	n/a	n/a	●◑
RWRQ15 Distributed function execution for INS							
Peer-to-peer	○	○	○	n/a	n/a	n/a	○
Client-to-server	●◑	●◑	●◑	n/a	n/a	n/a	●◑
Server-to-client	○	○	○	n/a	n/a	n/a	○

Continued on next page

Tab. B.5.: Secure Interactive Services: Assessment of Related Works (continued)

	Cheng et al. [Che14]	Howson et al. [How+11]	Franke et al. [Fra+14]	Ghiglieri et al. [GT14]	Michele et al. [MK14]	Lee et al. [Lee+13]	Mikityuk
RWRQ16 Distributed execution within a single function for INS	●◑	●◑	●◑	n/a	n/a	n/a	●
RWRQ17 Security of RSE for INS	○	○	○	n/a	n/a	n/a	●◑
RWRQ18 Third-party openness of INS	○	○	●◑	n/a	n/a	n/a	◐
RWRQ19 Privacy of INS	○	○	○	●◑	n/a	n/a	○
RWRQ20 Security of INS	○	○	○	●◑	n/a	n/a	○
RWRQ21 Device security of INS	○	○	○	◐	●◑	●◑	●◑