Briser la confiance : dissection et analyse des impacts sécurité et vie privée du DRM widevine

Gwendal Patat

▶ **To cite this version:**

Gwendal Patat. Briser la confiance : dissection et analyse des impacts sécurité et vie privée du DRM widevine. Cryptographie et sécurité [cs.CR]. Université de Rennes, 2023. Français. NNT : 2023URENS070 . tel-04446310

HAL Id: tel-04446310
https://theses.hal.science/tel-04446310

Submitted on 8 Feb 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

COLLEGE DOCTORAL BRETAGNE

MATHS, TELECOMS
INFORMATIQUE, SIGNAL
SYSTEMES, ELECTRONIQUE

Université de Rennes

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE RENNES

ÉCOLE DOCTORALE N° 601
Mathématiques, Télécommunications, Informatique, Signal, Systémes, Electronique
Spécialité : *Informatique*

Par
Gwendal PATAT

Bust the Trust : Dissect and Analyze the Security and Privacy Impacts of the Widevine DRM

Thèse présentée et soutenue à IRISA, Rennes, le 15 décembre 2023
Unité de recherche : UMR 6074

Rapporteurs avant soutenance :

Davide BALZAROTTI	Professeur, Eurecom, France
Jiska CLASSEN	Researcher, Hasso Plattner Institut, Allemagne
Romain ROUVOY	Professeur, Université de Lille, France

Composition du Jury :

Président :	Thomas JENSEN	Directeur de Recherche, INRIA, France
Examinateur·rices :	Davide BALZAROTTI	Professeur, Eurecom, France
	Jiska CLASSEN	Researcher, Hasso Plattner Institut, Allemagne
	Harry HALPIN	Research Scientist, MIT, Etats-Unis d'Amérique
	Romain ROUVOY	Professeur, Université de Lille, France
Dir. de thèse :	Pierre-Alain FOUQUE	Professeur, Université de Rennes, France
Encadrant :	Mohamed SABT	Maître de Conférence, Université de Rennes, France

When your mom hands you a note to miss school, the first thing you do is, you fold it and you put it in your pocket. I mean, if it's real, where's the crease?

Frank Abagnale Jr. – Catch me if you can

À ma maman,
pour m'avoir appris à ne pas être raisonnable.

Remerciements

[Only available in the printed version.]

Contents

Résumé en Français

L'omniprésence des contenus médiatiques en ligne a fondamentalement remodelé la façon dont nous consommons de l'information et des divertissements. Des plateformes de streaming aux e-books, notre accès au contenu est médiatisé par des infrastructures numériques conçues pour assurer à la fois la diffusion et la protection des matériaux soumis au droit d'auteur. Les jours où la possession de médias était la norme sont révolus. Aujourd'hui, les services Over-the-Top (OTT) comme Netflix, Disney+ ou Amazon Prime sont devenus pour beaucoup le moyen principal de regarder des vidéos. Ces plateformes distribuent du contenu via Internet, principalement via des modèles basés sur l'abonnement. Sous ce modèle, les utilisateurs peuvent accéder à des médias aussi souvent qu'ils le souhaitent ; cependant, le contenu devient inaccessible une fois l'abonnement terminé ou si le service de streaming ne le propose plus.

Le volume énorme de la distribution de ces médias, surtout dans des environnements qui pourraient être considérés comme non fiables, comme des appareils obsoletes, présente des défis significatifs pour les créateurs de contenu. Une préoccupation majeure parmi celles-ci est le piratage. Pour mettre cela en perspective, en 2020, la taille du marché OTT était estimée à 13,9 milliards de dollars et devrait grimper à un impressionnant 139 milliards de dollars d'ici 2028 [For22a].

Pour contrer ces défis, les fournisseurs de contenu s'appuient fortement sur les technologies Digital Rights Management (DRM). Les systèmes DRM visent à protéger les médias contre la redistribution non autorisée. Ils garantissent que le contenu est délivré sous forme chiffré, pour être déchiffré uniquement lorsqu'il est approuvé sur des appareils divers allant des smartphones aux Smart TV, en passant par les casques VR. Le défi auquel sont confrontés les développeurs de DRM et les créateurs de contenu est unique en ce sens que, dans ce contexte spécifique, l'attaquant peut en effet être un utilisateur légitime. Contrairement aux menaces de sécurité traditionnelles, cet utilisateur ne cherche pas principalement à accéder gratuitement aux actifs ; au lieu de cela, ils visent à contourner la protection DRM pour redistribuer le contenu sans restrictions. Ce qui distingue ce type d'attaquant dans le domaine de la sécurité informatique, ce sont leurs capacités accrues : ils sont des utilisateurs légitimes du service et possèdent des droits administratifs complets sur leur appareil, leur permettant de modifier les programmes s'éxecutant et d'effectuer une surveillance en temps réel de l'état de la machine. Pour contrer ces capacités accrues d'utilisateurs légitimes devenus attaquants, les solutions DRM ont historiquement opté pour des implémentations à code source fermé qui s'appuient fortement sur des techniques d'obfuscation pour protéger les clefs secrètes, ainsi que des techniques de traçage pour trouver l'origine d'une fuite. Ces stratégies visent à ralentir le processus de rétro-ingéniérie, réduisant ainsi le risque de compromission du système, et permettant d'entreprendre des actions légales contre les pirates. En gardant les détails d'implémentation opaque, les développeurs de DRM cherchent à créer une couche

supplémentaire de complexité afin de décourager les attaquants et entraver la redistribution non autorisée.

Les systèmes DRM ne sont pas un concept nouveau ; ils existaient bien avant l'essor des médias distribués sur Internet. Historiquement, la technologie la plus ancienne étroitement liée au DRM est généralement attribuée à Ryuichi Moriya, ayant breveté le Software Service System (SSS) en 1983 [MT88]. Cette approche, appelée superdistribution, proposait le chiffrement des logiciels et contenus numériques tout en permettant leur redistribution gratuite dans leur forme protégée. Les consommateurs n'étaient tenus d'acheter le bien que lorsqu'ils souhaitaient utiliser le produit, à quel moment les clés de déchiffrement seraient fournies. Pour les supports physiques, le DRM Content Scramble System (CSS) [DVD] a été popularisé avec l'émergence des DVD permettant le déchiffrement des médias uniquement sur des applications autorisées, remplacées plus tard en 2005 par la norme Advanced Access Content System (AACS) pour les HD DVD et les disques Blu-ray [Adv]. Au début des années 2000, avec l'augmentation de la consommation de médias sur des ordinateurs personnels et des appareils portables tels que les iPods, Microsoft et Apple ont tous deux publié leurs propres systèmes DRM, respectivement Microsoft Windows Media DRM (WMDRM) [Micb] et Apple FairPlay [App23]. Juste quelques années après, le PDG d'Apple, Steve Jobs écrivit sa célèbre lettre ouverte "Thoughts on Music" [Ste07], mettant en perspective les avantages de l'utilisation du DRM, afin pour rassurer les créateurs de contenu et l'industrie de la musique, tout en échouant à endiguer le piratage. Ce raisonnement mena par la suite à la fin de l'utilisation des DRMs pour iTunes et à l'abandon d'Apple FairPlay en 2009.

Cependant, à partir des années 2010, le paysage des DRM a connu une transition significative alors que ces systèmes de DRM traditionnels se sont adaptés pour répondre aux exigences de l'industrie du streaming en plein essor. La prolifération de diverses plateformes de streaming nécessitait des solutions DRM flexibles pouvant gérer un large éventail de formes de médias et de permissions d'accès. La nature changeante de la consommation de contenu, n'étant plus un achat unique mais souvent sous forme d'abonnement, a également nécessité des changements dans le fonctionnement des systèmes de DRM. Les mécanismes internes des DRM ont changé, et avec au fil des versions, WMDRM a été déprécié pour Microsoft PlayReady [Mic23], Apple FairPlay a été relancé pour devenir Apple FairPlay Streaming (FPS), et de nouveaux acteurs ont rejoint la course comme Adobe Primetime [Ado], Marlin DRM [Mara], ou Widevine [Goo23d]. Cette évolution de la technologie DRM est donc devenue étroitement liée à la croissance de l'industrie du streaming, façonnant la manière dont le contenu est accessible et consommé.

Avec cette croissance, et compte tenu de la montée de la consommation de médias sur les appareils mobiles, Android est devenue une plateforme clef pour le contenu OTT. Reconnaissant ce changement, de nombreuses plateformes OTT ont développé des applications spécifiques à Android, ciblant un public large dans un business en plein essor. En réponse, Android s'est doté d'implémentation d'interface DRM, facilitant l'intégration de systèmes propriétaires permettant le déchiffrement de flux médias protégés. Bien que ce cadre soit compatible avec divers systèmes DRM, le choix du DRM dépend souvent du fabricant en raison de la présence de clients DRM natifs dans les implémentations des fournisseurs. Parmi ceux-ci, Widevine, le système DRM de Google, se distingue en raison de son déploiement généralisé, non seulement sur les appareils Android mais également sur les SmartTV et les navigateurs Web tels que Chrome ou Firefox. Cependant, comme pour toutes les technologies, son déploiement soulève des questions pertinentes sur son efficacité opérationnelle, sa robustesse en matière de sécurité et ses implications pour la vie privée des utilisateurs. En effet, la route vers l'adoption des DRM

a été critiquée, beaucoup soulignant les insécurités perçues dans ces systèmes. Néanmoins, la montée en puissance des plateformes OTT a poussé l'adoption et l'affinement continus des DRM. Soulignant cette tendance, le World Wide Web Consortium (W3C), malgré les critiques qu'il a dû affronter [Hal17], a introduit Encrypted Media Extension (EME) [DSWB19], en la marquant comme la première norme Web officielle pour les DRM.

Malgré ces préoccupations, peu de recherches publiques ont été menées sur les systèmes de DRM actuellement déployés. La raison principale derrière un tel manque d'analyse de sécurité est que la clause 1201 du Digital Millennium Copyright Act (DMCA) rend illégal l'étude des systèmes de DRM. Le résultat est qu'en vertu du DMCA, les chercheurs ne peuvent pas enquêter sur les vulnérabilités de sécurité si cela nécessite de la rétro-ingéniérie. Cette loi a déjà été utilisée contre des chercheurs pour censurer leurs travaux, comme l'a montré Hewlett-Packard contre Snosoft en 2002 [McC02]. Malheureusement, d'autres cas ont suivi, et plus de cinquante affaires judiciaires ont été lancées contre des chercheurs jusqu'en 2016 [Ele17].

Heureusement, les restrictions ont été partiellement levées récemment. En effet, en octobre 2018, l'American Library of Congress et le Copyright Office ont élargi les exemptions à la clause 1201 du DMCA. Par conséquent, en théorie, les chercheurs en sécurité peuvent maintenant librement enquêter, corriger et publier des failles de sécurité sur les solutions DRM. Il en va de même en Europe avec l'équivalent du DMCA, l'European Union Copyright Directive (EUCD).

En raison de ces changements juridiques, notre recherche se présente comme l'une des premières études complètes dans ce domaine de nouveau accessible. Nous profitons de l'opportunité offerte par ces exemptions pour examiner les systèmes de DRM qui étaient auparavant impénétrables par la recherche publique.

Cependant, ces exemptions n'ont pas empêché Google qui, en novembre 2020, a supprimé tous les dépôts GitHub incluant les clés secrètes de Widevine. Cela ne signifie pas que les exploits compromettant Widevine n'ont jamais été publiés auparavant. En effet, la base de données Common Vulnerabilities and Exposures (CVE) du MITRE [MIT22] répertorie 25 enregistrements CVE depuis 2014, expliquant différentes problématiques de sécurité au sein des implémentations de Widevine. Malgré un tel dossier public, il n'y a pas beaucoup de littérature fournissant des informations approfondies sur la sécurité de Widevine.

Cette thèse entreprend une étude complète du principal acteur DRM sur le marché, à savoir Widevine. L'objectif est de comprendre ses mécanismes de protection, d'identifier d'éventuelles vulnérabilités et d'évaluer son impact sur la vie privée des utilisateurs. À une époque où les interactions numériques sont omniprésentes et où les préoccupations concernant la sécurité et la vie privée s'intensifient, il devient impératif d'examiner des systèmes opaques comme Widevine qui jouent un rôle prédominant dans la consommation de contenu numérique.

Avec cette thèse, nous aspirons à présenter une analyse approfondie de Widevine, mettant en lumière ses forces et ses faiblesses, ainsi que ses implications plus larges pour les créateurs et les consommateurs de contenu. Cette entreprise cherche à combler le fossé entre les subtilités techniques et la compréhension des utilisateurs, fournissant une perspective plus claire sur le domaine.

1. Contributions

Dans cette thèse, nous examinons le DRM Widevine afin d'étudier ses implications en matière de sécurité et de confidentialité pour les entreprises, telles que les plateformes OTT, ainsi que pour les utilisateurs. La recherche publiquement disponible sur les systèmes déployés a été lacunaire dans ce domaine pendant près d'une décennie. Avec notre travail, nous avons l'intention de combler ce vide en analysant l'acteur dominant dans ce domaine en posant trois questions de recherche principales auquels nous répondrons tout au long de nos contributions :

I. *Quel est l'impact de la nature opaque de Widevine sur la recherche de vulnérabilités, et qu'est-ce que cela implique pour la sécurité globale du contenu protégé ?*

II. *Comment Widevine est-il utilisé par les plateformes OTT compte tenu de ses implémentations opaques ?*

III. *De quelles manières l'implémentation propriétaire fermée de Widevine influence-t-elle la vie privé des utilisateurs, particulièrement en termes de web tracking ?*

Les contributions de cette thèse sont divisées en trois catégories. Premièrement, nous analysons le fonctionnement interne du système DRM Widevine au sein de l'écosystème Android pour obtenir une vue globale du protocole et de ses opérations cryptographiques. En faisant cela, nous examinons la confidentialité du contenu, qui est au cœur des objectifs de Widevine. Dans cette partie, nous avons démêlé ses composants internes et trouvé des problèmes de sécurité dans les implémentations et l'utilisation de Widevine par les OTT. Deuxièmement, le statut de Widevine en tant que système fermé et propriétaire complique la vérification indépendante de son comportement, posant des défis pour les chercheurs et les utilisateurs préoccupés par la confidentialité et la sécurité. Ce manque de transparence empêche de confirmer si le système fonctionne comme prévu. En effet, en s'appuyant sur des primitives cryptographiques, le protocole propriétaire de Widevine incorpore également des identifiants uniques que les sites Web malveillants pourraient abuser pour suivre les utilisateurs en ligne. Nous démontrons que de telles préoccupations sont justifiées en examinant l'impact sur la vie privée des utilisateurs qu'un OTT curieux ou un site malveillant pourrait avoir. Enfin, l'évaluation de la sécurité de Widevine repose sur une compréhension approfondie et complexe de son protocole. Dans la dernière partie de nos contributions, nous proposons une représentation préliminaire du protocole à l'aide de l'outil de vérification Tamarin ainsi que des buts communs au DRM concernant la sécurité des medias afin de vérifier formellement ces objectifs de sécurité.

1.1. Analyse de la sécurité de Widevine

L'une des contributions de cette thèse est l'ingénierie inverse complète du protocole Widevine non documenté et des binaires.

Fonctionnement interne de Widevine. Comme première approche des systèmes DRM, nous présentons le protocole Widevine étant non documenté et à source fermé ainsi que ses composants cryptographiques. Nous documentons la logique derrière son échelle de clés et ses phases de provisionnement grâce à la rétro-ingéniérie du Content Decryption Module (CDM) Widevine sur Android. Sur la base de ces connaissances acquises, nous avons conçu

WideXtractor, un outil analysant le flux d'éxecution du protocole ainsi que tous les échanges de messages entre le clients et les serveurs distants. Nous montrons l'efficacité de WideXtractor en inspectant l'utilisation de Widevine par Netflix, révélant ainsi une utilisation non standard de Widevine. De plus, nous montrons comment récupérer facilement la Root-of-Trust (RoT) L3 de Widevine, permettant aux attaquants d'obtenir n'importe quel contenu sub-HD. Étant largement déployé, la nature opaque de Widevine et sa sécurité sont étudiés ici pour répondre à la question **I**.

> **Reference:** Ces travaux sont le résultat d'une collaboration avec Mohamed Sabt et Pierre-Alain Fouque. Ils furent publiés dans les actes du *IEEE Symposium on Security and Privacy Workshop On Offensive Technologies 2022* [PSF22a].

Utilisation de Widevine par les OTTs. En analysant l'utilisation de Widevine par les OTT dans la nature, nous présentons une évaluation empirique des mécanismes de protection utilisés par les fournisseurs de contenu au sein d'Android pour protéger leurs contenus multimédias afin de répondre à la question **II**. Notre étude montre que les applications OTT s'appuient sur Widevine en utilisant les Trusted Execution Environment (TEE) mobiles lorsqu'ils sont disponibles, tout en n'appliquant que partiellement les recommandations de Widevine concernant le chiffrement de la vidéo et de l'audio ainsi que sur la révocation des appareils. Sur ce dernier point, nous étudions le support des appareils Android obsolètes par les OTTs, ces apparails ne recevant plus de mises à jour de sécurité. Nous exposons que les applications OTTs augmentent la surface d'attaque des médias en privilégiant une large distribution du contenu au détriment de la sécurité. Nous montrons que de tels vecteurs peuvent être facilement exploités pour accéder aux contenus non protégés en présentant six applications OTT, y compris Netflix, Hulu et Showtime.

> **Reference:** Ces travaux sont le résultat d'une collaboration avec Mohamed Sabt et Pierre-Alain Fouque. Ils furent publiés dans les actes du *IEEE/IFIP International Conference on Dependable Systems and Networks 2022* [PSF22b].

1.2. Impact de Widevine sur la vie privé des utilisateurs

Lors de sa standardisation, EME a été décriée de part le fait qu'aucun contrôle ne pouvait être effectué sur les systèmes DRM sous-jacents afin de vérifier s'ils étaient conforme à la fois en termes de sécurité et de confidentialité. Ce travail montre que ces préoccupations en matière de confidentialité sont fondées en enquêtant sur le système DRM Widevine utilisable via l'API web EME. Nous présentons comment un site web curieux peut exploiter le protocole Widevine et une mauvaise implémentation au sein des navigateurs internet afin de recueillir des fingerprints uniques et stables à long terme pour tous les appareils mobiles Android ainsi qu'un User-Agent Header non modifiable pour Android et les OS desktop Windows, macOS et Linux. En plus de ces identifiants, nous exploitons un mécanisme avec état pouvant être abusé pour suivre les utilisateurs en ligne tel un supercookie. Nous fournissons des informations sur la conformité des implémentations actuelles des navigateurs considérant les recommandations sur la vie privé présentées par EME, montrant un impact réel tout en enquêtant sur la question **III**.

> **Reference:** Ces travaux sont le résultat d'une collaboration avec Mohamed Sabt et Pierre-Alain Fouque. Ils furent publiés dans les actes du *Privacy Enhancing Technologies Symposium 2023* [PSF23].

1.3. Vérification formelle du protocole Widevine EME

Dans le domaine des systèmes DRM, prouver des garanties de sécurité formelles est souvent un aspect négligé mais crucial. Bien que Widevine ait été largement déployé sur diverses plateformes, facilité par la standardisation de EME, la vérification formelle de son protocole doit encore être étudiée. Cette contribution vise à combler cette lacune en utilisant Tamarin [SMCB12 ; MSCB13], un outil de vérification formelle, pour traduire le protocole Widevine tel qu'utilisable par EME. Étant donné que ce travail est en cours, nous ne présentons pas de conclusions définitives mais délimitons des objectifs de sécurité clairs auquels tout système DRM devrait répondre. Nous présentons ces objectifs au sein de Tamarin sous forme de règles formelles. Tout au long de ce processus, nous présentons les problèmes rencontrés lors de l'élaboration de cette représentation afin de décrire pleinement le fonctionnement interne de Widevine et suggérons des solutions à ces problèmes. Cette étude marque une première étape vers la compréhension formelle de la sécurité des systèmes DRM et répond à des questions non explorées en rapport avec leurs challenges de sécurité.

> **Reference:** Ces travaux sont issues d'une collaboration en cours avec Stéphanie Delaune, Joseph Lallemand, Florian Roudot, et Mohamed Sabt.

2. Artéfacts et Impacts

Nos contributions sont multiples et un effort particulier a été mis en place afin que nos résultats puissent être reproduits et utilisés dans des recherches futures. Pour faciliter cela, nous avons rendu disponibles des artefacts au sein de dépôts publiques GitHub. Dans les sections suivantes, nous énumérons les résultats issues de nos contributions.

Sécurité de Widevine. Dans la sous-section 1.1, nous avons présenté deux conséquences liées au système opaque de Widevine : une vulnérabilité menant à la récupération complète des médias en clair ainsi qu'une utilisation non optimale de Widevine par les OTT en ce qui concerne la protection des actifs. Ces impacts sont illustrés par nos outils et Proof-of-Concept (PoC) :

- Notre outil WideXtractor, disponible à `https://github.com/Avalonswanderer/wideXtractor`, peut être utilisé pour tracer l'exécution de Widevine sur Android ainsi que desktop et récupérer tous les messages échangés. Cet outil peut ensuite être utilisé avec notre script `https://github.com/Avalonswanderer/widevine_key_ladder` afin d'imiter la key ladder propriétaire de Widevine.

- Notre PoC complet pour la récupération des clés Widevine L3 peut être trouvé à `https://github.com/Avalonswanderer/widevinel3_Android_PoC`.

Ces problème ont été divulgués aux parties concernées et a conduit à un correctif de la librarie cryptographique lié à l'Original Equipment Manufacturer (OEM) `liboemcrypto`

https://source.android.com/docs/security/bulletin/2021-08-01#widevine, et à mené à la CVE-2021-0639,[1]. Ces impacts nous ont permis d'être reconnus au sein du Hall of Fame de Google et Netflix.

Confidentialité de Widevine. Dans la sous-section 1.2, nous décrivons notre analyse sur l'impact relatif à la vie privé de Widevine pour les utilisateurs lorsqu'il est utilisé via l'API EME des navigateurs Web. Notre travail a abouti à un outil appelé EME Tracker, disponible à https://github.com/Avalonswanderer/widevine_eme_fingerprinting, démontrant comment un site Web curieux peut recueillir des identifiants uniques sur les utilisateurs afin de suivre leurs activités en ligne. Ce travail a été reconnu par le Mozilla Security Bug Bounty Program Hall of Fame, et amena à un correctif disponible depuis Mozilla Firefox 109.0a1.[2]

[1] https://www.cve.org/CVERecord?id=CVE-2021-0639

[2] https://hg.mozilla.org/mozilla-central/rev/73a5383507573d87efb3daf896334e425a0620b6

Publications

In this chapter, we list peer-reviewed publications in the proceedings of international conferences in chronological order. This thesis is built on the basis of these publications and some ongoing work which have not been peer-reviewed yet.

Publication I	[PSF22a]	*Exploring Widevine for Fun and Profit*, Gwendal Patat, Mohamed Sabt and Pierre-Alain Fouque. Published in the proceedings of IEEE S&P, WOOT 22 (p.277-288)
Publication II	[PSF22b]	*WideLeak: How Over-the-Top Platforms Fail in Android*, Gwendal Patat, Mohamed Sabt and Pierre-Alain Fouque. Published in the proceedings of DSN 2022 (p.501-508)
Publication III	[PSF23]	*Your DRM Can Watch You Too: Exploring the Privacy Implications of Browsers (mis)Implementations of Widevine EME*, Gwendal Patat, Mohamed Sabt and Pierre-Alain Fouque. Published in the proceedings of PETS 2023 (p.306-321)

Abbreviations and Notations

AACS Advanced Access Content System.

AACS LA Advanced Access Content System Licensing Administrator.

AES Advanced Encryption Standard.

API Application Programming Interface.

CBC Cipher Block Chaining.

CDM Content Decryption Module.

CDN Content Delivery Network.

CENC Common Encryption Scheme.

CRC Cyclic Redundancy Check.

CSS Content Scramble System.

CVE Common Vulnerabilities and Exposures.

DASH Dynamic Adaptative Steaming over HTTP.

DMCA Digital Millennium Copyright Act.

DRM Digital Rights Management.

DVD CCA DVD Copy Control Association.

ECP Enhanced Content Protection.

EME Encrypted Media Extension.

ESN Equipment Serial Number.

EUCD European Union Copyright Directive.

FIFO First In, First Out.

FPS FairPlay Streaming.

GDPR General Data Protection Regulation.

HAL Hardware Abstraction Layer.

HD High Definition.

HLS HTTP Live Streaming.

IV Initialization Vector.

JNI Java Native Interface.

KCB Key Control Block.

L1 Widevine security Level 1.

L2 Widevine security Level 2.

L3 Widevine security Level 3.

MBB Marlin Broadband.

MDC Marlin Development Community.

MPD Media Presentation Description.

MPEG Moving Picture Experts Group.

MS3 Marlin Simple Secure Streaming.

MSL Message Security Layer.

MTMO Marlin Trust Management Organization.

NDA Non-Disclosure Agreement.

OEM Original Equipment Manufacturer.

OMA Open Mobile Alliance.

OS Operating System.

OTT Over-the-Top.

PoC Proof-of-Concept.

QSEE Qualcomm Secure Execution Environment.

RoT Root-of-Trust.

SCE Secure Content Exchange.

SFS Secure File System.

SoC System-on-Chip.

SRM Secure Removable Media.

SSS Software Service System.

STB Set-Top Box.

SVP Secure Video Path.

TEE Trusted Execution Environment.

TTL Time-To-Live.

UUID Universal Unique Identifier.

VMP Verified Media Path.

VRP Vulnerability Reward Program.

W3C World Wide Web Consortium.

WMDRM Windows Media DRM.

Introduction

1 Introduction

This is my favorite book in all the world, though I have never read it.

Princess Bride

The ubiquity of online media content has fundamentally reshaped how we consume information, entertainment, and knowledge. From streaming platforms to e-books, our access to content is mediated by digital infrastructures designed to ensure both the dissemination and protection of copyrighted materials. Gone are the days when owning media was the norm. Today, OTT, such as Netflix, Disney+, or Amazon Prime, have become the primary channels for many to watch videos. These platforms distribute multimedia content over the Internet, primarily through subscription-based models. Under this model, users can access and play media as often as they desire; however, the content becomes inaccessible once the subscription ends or if the streaming service no longer offers it.

The sheer volume of media distribution, especially in environments that may be deemed untrusted, like specific devices, introduces significant challenges for content creators. A primary concern among these is piracy. To put this into perspective, in 2020, the OTT market size was valued at $13.9 billion and is projected to escalate to an impressive $139 billion by 2028 [For22a].

To counter these challenges, content providers heavily lean on DRM technologies. DRM systems aim to protect media from unauthorized distribution. They ensure that content is delivered in encrypted formats, only to be decrypted when approved on user devices ranging from smartphones to Smart TVs, including VR headsets. The challenge facing DRM developers and content creators is unique in that, in this specific context, the attacker may actually be a legitimate user. Unlike traditional security threats, this user is not primarily seeking free access to assets; instead, they aim to bypass DRM protection to redistribute content without restrictions. What sets this type of attacker apart in the field of computer security is their elevated capabilities: they are legitimate users of the service and possess full administrative rights to their device, enabling them to modify code and perform real-time monitoring. To counteract these enhanced capabilities of legitimate users-turned-attackers, DRM solutions have historically opted for closed-source implementations that heavily rely on obfuscation techniques to protect secret, and traitor tracking techniques to find the origin of a leak. These strategies aim to slow the reverse engineering process, thereby reducing the risk of system compromise, and enabling legal action to be undertaken. By keeping the implementation details concealed, DRM developers seek to create an additional layer of complexity that deters tampering and unauthorized distribution.

DRM systems are not a novel concept; they existed well before the rise of Internet-based media. Historically, the earliest technology closely related to DRM is generally credited to Ryuichi Moriya, who patented the SSS in 1983 [MT88]. This approach, called superdistribution,

proposed the encryption of digital assets while still permitting their redistribution free of charge in their protected form. Consumers were only required to purchase the good when they used the product, at which point the decryption keys would be provided. For physical media, the CSS DRM [DVD] has been popularized with the emergence of DVDs allowing decryption only on licensed applications, later replaced in 2005 by the AACS standard for HD DVD and Blu-ray Disc [Adv]. In early 2000, with the rise of media consumption on personal computers and portable devices such as iPods, Microsoft and Apple both released their own DRM systems, respectively Microsoft WMDRM [Micb] and Apple FairPlay [App23]. Just a few years after these releases, Apple CEO Steve Jobs wrote his now famous *"Thoughts on Music"* [Ste07], putting in perspective the benefits of using DRM to reassure content creators and the music industry that still fail to circumvent piracy. This later led to the end of DRM for iTunes and the deprecation of Apple FairPlay in 2009.

However, starting in the 2010s, the DRM landscape saw a significant transition as these traditional DRM systems adapted to meet the requirements of the booming streaming industry. The proliferation of various streaming platforms required flexible DRM solutions that could handle a wide range of media forms and access permissions. The changing nature of how content was consumed, being no longer a single purchase but often part of a subscription model, also necessitated changes in how DRM systems functioned. DRMs inner workings have changed, and with versions, WMDRM has been deprecated for Microsoft PlayReady [Mic23], Apple FairPlay revived to become Apple FairPlay Streaming (FPS), and new players have joined the race with Adobe Primetime [Ado], Marlin DRM [Mara], or Widevine [Goo23d]. This evolution in DRM technology has thus become intricately connected with the streaming industry growth, shaping how content is accessed and consumed.

With this growth, and given the surge in media consumption on mobile devices, Android has emerged as a key platform for OTT content. Recognizing this shift, numerous OTT platforms have developed Android-specific apps, targeting an expansive audience in a booming business environment. As a response, Android has equipped itself with a DRM framework, facilitating the decryption of protected media streams. Although this framework is compatible with various DRM systems, the choice of DRM often depends on device manufacturers due to the presence of native DRM clients within vendor codes. Among these, Widevine, Google DRM system, stands out due to its widespread deployment, not just on Android devices but also on SmartTV and web browsers like Chrome or Firefox. However, as with all technologies, its deployment raises pertinent questions about its operational efficacy, security robustness, and implications for user privacy. Indeed, the road to DRM adoption was criticized, with many pointing out perceived insecurities in such systems. Regardless, the increasing prominence of OTT platforms has pushed the continuous adoption and refinement of DRMs. Highlighting this trend, the W3C, despite facing critiques [Hal17], introduced the EME [DSWB19], marking it as the first official web standard for DRM.

Despite concerns, little public research has been conducted on currently deployed DRM systems. The main reason behind such a lack of public security analysis is that the DMCA's 1201 clause makes it illegal to study DRM systems. The result is that, under the DMCA, researchers cannot investigate security vulnerabilities if doing so requires reverse engineering. This law has already been used against security researchers to censor their work, as shown by Hewlett-Packard against Snosoft in 2002 [McC02]. Unfortunately, more cases have followed, and over fifty court cases have been launched against researchers as of 2016 [Ele17].

Fortunately, restrictions have been partially lifted recently. Indeed, in October 2018, the American Library of Congress and the Copyright Office expanded the exemptions to the

DMCA's 1201 clause. Consequently, in theory, security researchers can now freely investigate, correct, and publish security flaws on DRM solutions. The same has been done in Europe with the DMCA equivalent, the EUCD.

Due to these legal changes, our research stands as one of the first comprehensive studies in this newly accessible domain. We leverage the opportunity provided by these exemptions to examine DRM systems that were previously impenetrable for public research.

However, such exemptions did not stop Google that, in November 2020, took down all GitHub repositories including secret keys of Widevine. That does not mean that exploits compromising Widevine have never been published before. Indeed, the MITRE CVE database [MIT22] lists 25 CVE records since 2014, explaining different security issues within Widevine implementations. Despite such a public record, there is not much literature providing deep insights about Widevine security.

This thesis undertakes a comprehensive study of the leading DRM actor, namely Widevine. The aim is to understand its mechanisms for content protection, identify potential vulnerabilities, and assess its impact on user privacy. In an age where digital interactions are ubiquitous, and concerns about security and privacy are escalating, it becomes imperative to scrutinize systems like Widevine that play a prominent role in digital content consumption.

With this thesis, we aspire to present an in-depth analysis of Widevine, shedding light on its strengths and weaknesses, and its broader implications for content creators and consumers. This endeavor seeks to bridge the gap between technical intricacies and user understanding, providing a clearer perspective on the area.

1.1. Contributions of this Thesis

In this thesis, we look at the Widevine DRM to investigate its security and privacy implications for businesses, such as OTT, and end users. Publicly available research on deployed systems has been lacking in this area for almost a decade. With our work, we intend to fill the gap by analyzing the current top player in the domain by putting in front three main research questions that we try to answer throughout our contributions:

I. *What is the impact of the opaque nature of Widevine regarding vulnerability discovery, and what are the repercussions on the overall security of protected media content?*

II. *How does Widevine is leveraged by OTT platforms considering its opaque implementations?*

III. *In what ways does the proprietary and closed-source Widevine protocol influence privacy concerns for end users, particularly in terms of online tracking?*

Contributions of this thesis are separated into three categories. First, we analyze the inner workings of the Widevine DRM system within the Android ecosystem to grasp an expansive view of the protocol and its cryptographic operations. In doing so, we investigate content confidentiality, which is the core of Widevine goals. In this part, we unraveled its inner components and found security issues within implementations and Widevine usage by OTTs. Second, Widevine status as a closed-source, proprietary system complicates independent verification of its behavior, posing challenges for both researchers and users concerned about privacy and security. This lack of transparency hinders confirming whether the system

operates as intended. Indeed, by relying on cryptographic primitives, Widevine proprietary protocol also encompasses unique identifiers that malicious websites can misuse to track users. We demonstrate that such concerns are justified by inspecting the degree of privacy leakage a curious OTT or website could gather from a user and find a unique identifier in cleartext. Finally, Widevine security can take much work to apprehend due to its complexity and workflow. In the last part of our contributions, we propose an early representation of the protocol using the Tamarin verification tool to verify DRM systems security goals formally.

1.1.1. Analysis of Widevine Security

One of the contributions of this thesis is the full reverse engineering of the undocumented Widevine protocol and binaries.

Widevine Core. As a first approach to DRM systems, we present the undocumented closed-source Widevine protocol with its cryptographic components. We extract the logic behind its key ladder and provisioning phases by reverse engineering the Widevine CDM on Android. Based on the gained insights, we design WideXtractor, a tool analyzing the protocol workflow and all message exchanges between clients and distant servers. We show the effectiveness of WideXtractor by inspecting the use of Widevine by Netflix, thereby uncovering interesting findings about Netflix asset protection. Furthermore, we can trivially recover the L3 RoT, which allows attackers to obtain any content of sub-HD quality. Being widely deployed, its opaque nature and risk are studied here to answer question **I**.

> **Reference:** This work is the outcome of a joint work with Mohamed Sabt, and Pierre-Alain Fouque. It has been published in the proceedings of *IEEE Symposium on Security and Privacy Workshop On Offensive Technologies 2022* [PSF22a].

The Case of OTTs Usage. By analyzing the usage of Widevine by OTTs in the wild, we report on an empirical evaluation of the protection mechanisms used by content providers within Android to protect their media assets to answer question **II**. Our study shows that OTT apps rely on Widevine using TEE protection when available but do not comply with Widevine recommendations in encrypting video and audio with distinct keys or regarding revocation of old devices. We investigate OTT app support of Android legacy devices running the software-only Widevine DRM no longer receiving security updates. We expose that OTT apps have extended the device attack surface by choosing large content distribution over security. We show that such vectors can be trivially exploited to recover DRM-free movies from six OTT apps, including Netflix, Hulu, and Showtime.

> **Reference:** This work is the outcome of a joint work with Mohamed Sabt, and Pierre-Alain Fouque. It has been published in the proceedings of *IEEE/IFIP International Conference on Dependable Systems and Networks 2022* [PSF22b].

1.1.2. Widevine Privacy Leakage Investigation

During its standardization, EME was decried because no effective control can be performed on the underlying closed-source DRM systems to check whether they comply both in terms of security and privacy. This work shows that these privacy concerns are founded by investigating the closed-source Widevine DRM system through the EME Web API. We present how a

curious origin can leverage the Widevine protocol and misimplementation within browsers to gather unique and long-term stability fingerprints for Android mobile devices and a never-lying User-Agent header for both mobile and desktop. In addition to this stateless tracking, we exploit a stateful mechanism that websites can misuse to track users online. We provide insights on current user-agent implementations compliance with EME privacy guidelines, showing a real impact while investigating question **III**.

> **Reference:** This work is the outcome of a joint work with Mohamed Sabt, and Pierre-Alain Fouque. It has been published in the proceedings of *Privacy Enhancing Technologies Symposium 2023* [PSF23].

1.1.3. Formal Analysis of the Widevine EME Protocol

In the realm of DRM systems, ensuring formal security guarantees is often an overlooked yet crucial aspect. While Widevine has been extensively deployed across various platforms, deployment facilitated by EME, the formal verification of its protocol still needs to be studied. This contribution aims to fill this void by employing the Tamarin prover [SMCB12; MSCB13], a formal verification tool, to translate the Widevine protocol within the EME context. Given that this work is ongoing, we have yet to present definitive conclusions but have delineated clear security goals that any DRM system should strive for. We present these goals within Tamarin as formal rules. Throughout this process, we discover limitations in Tamarin capabilities to represent the inner workings of Widevine fully and suggest solutions to these issues. This exploratory study serves as a cornerstone for future research, aiming to rigorously verify the security of DRM systems and contribute to decreasing blind trust towards them.

> **Reference:** This work is an ongoing collaboration with Stéphanie Delaune, Joseph Lallemand, Florian Roudot, and Mohamed Sabt.

1.2. Artifacts and Impacts

Our contributions are manifold, and a significant emphasis has been placed on ensuring that our results can be reproduced with precision. To facilitate this, we have made available artifacts in distinct repositories. In the subsequent sections, we enumerate the outputs derived from our work across various domains.

Widevine Security. In subsection 1.1.1, we presented two consequences of the Widevine opaque system: a vulnerability leading to full media recovery and a misusage of Widevine by OTTs regarding assets protection. These impacts are illustrated by our tools and PoC:

- Our tool WideXtractor available at `https://github.com/Avalonswanderer/wideXtractor`, can be used to trace Widevine execution on Android and desktop to recover exchange buffers. This tool can then be used with our key ladder script at `https://github.com/Avalonswanderer/widevine_key_ladder` to mimic the proprietary Widevine key ladder.
- Our full Widevine L3 key recovery PoC can be found `https://github.com/Avalonswanderer/widevinel3_Android_PoC`.

This issue has been disclosed to the concerned parties and has led to a patch within the Android OEM Crypto library path `https://source.android.com/docs/security/bulletin/`

2021-08-01#widevine, and the CVE-2021-0639,[1] and allowed us to be acknowledged within Google and Netflix Hall of Fame.

Widevine Privacy. In subsection 1.1.2, we describe our analysis of Widevine privacy implication for users when used through the EME API of web browsers. Our work has resulted in a tool called EME Tracker, available at https://github.com/Avalonswanderer/widevine_eme_fingerprinting, demonstrating how a curious website can gather unique identifiers on users to track their activities online. This work has been acknowledged by the Mozilla Security Bug Bounty Program Hall of Fame, and the issue has been patched starting from Mozilla Firefox 109.0a1.[2]

1.3. Outline

This thesis is divided into three parts, excluding this introduction. Part I introduces some background on key notions that will be central to most contributions:

- Chapter 2 gives an overview of the DRM systems history and briefly presents current main actors.
- Chapter 3 describes the Android ecosystem for media playback and its DRM architecture allowing license acquisition and media decryption.
- Chapter 4 details EME from the W3C, the common API for web browsers to communicate with the fragmented DRM ecosystem.

Next, the core of the manuscript comes down to Part II, detailing all our contributions:

- Chapter 5 provides our first approach to the Widevine protocol and its internals. We present our revere engineering efforts and how we broke the DRM scheme for protected assests.
- Chapter 6 presents the first experimental study on the state of Widevine use in the wild. Our study explores OTT compliance with Widevine guidelines regarding asset protection and legacy phone support.
- Chapter 7 invastigates privacy leakage caused by EME relying the proprietary and closed-source Widevine DRM system. We conduct empirical experiments to show that browsers diverge when complying EME privacy guidelines, which might undermine users privacy.
- Chapter 8 proposes a formal security analysis of the Widevine used through the EME web API. Using Tamarin, we propose a protocol model representing our full understanding of this closed-source system.

Finally, Part III concludes and gives some perspectives on future works.

[1] https://www.cve.org/CVERecord?id=CVE-2021-0639
[2] https://hg.mozilla.org/mozilla-central/rev/73a5383507573d87efb3daf896334e425a0620b6

How to read. At the onset of each chapter, readers are provided with a concise overview, complemented by a *Takeaway box* that underscores the main themes and contributions. In Part II, each chapter initiates with a reference to the pertinent background. However, every chapter is designed to be self-sufficient, with essential concepts briefly revisited in *Reminder boxes*.

Reminder: This is a reminder box.

Takeaway boxes encapsulate the key points of technical section, allowing readers the option to either bypass the section without losing its essence or proceed through the section with a clear understanding of its objectives.

Takeaway: This is a takeaway box.

Part I.

Preliminaries

Digital Rights Management Systems 2

Love is sharing a password.

Netflix, before being Netflix.

This chapter offers an overview of Digital Rights Management (DRM) systems, highlighting the significant advancements in online video content protection since the 1980s. It briefly traces the historical development of DRM, emphasizing the technological shifts and challenges that have shaped the current landscape with a presentation of the leading DRM systems: Google Widevine, Microsoft PlayReady, and Apple FairPlay Streaming.

Together, these elements paint a concise picture of the DRM ecosystem, providing essential context for understanding its complexities and continued relevance in the online media industry.

Contents

2.1. DRM Evolution

2.1.1. Early Stage: The Advent of the Software Service System

In the formative years of digital content protection, one noteworthy approach was the Software Service System (SSS) [MT88]. Though rudimentary compared to later DRM schemes, this system laid the groundwork for the integration of software-based protection mechanisms in various digital platforms.

Software Service System (SSS). This system was among the initial attempts to manage and control access to digital resources through software-based methods. This method, known as superdistribution, involves encrypting digital assets while allowing their free redistribution in a secure format to promote sharing and increase potential consumers. Users are only obligated to make a purchase when they actually use the product, receiving the decryption keys at that juncture.

2.1.2. Physical DVDs and Blu-Ray Disc Era

This subsection delves into the DRM systems employed in the physical media formats of DVDs and Blu-Ray discs, focusing on the DVD Copy Control Association (DVD CCA)'s Content Scramble System (CSS) [DVD] and its spiritual successor, the Advanced Access Content System (AACS) [Adv].

DVD CCA and CSS. The DVD CCA introduced the CSS as an early DRM for DVDs in 1996. It employed a 40-bit stream cipher key, a restriction imposed by the American Export Administration Regulation [Exp] at the time. However, the system was quickly compromised in 1999 by a tool named DeCSS [lem00], developed by a group of three authors.

AACS. Designed to replace CSS in 2005, AACS used a significantly stronger 128-bit AES encryption algorithm. Despite its enhanced security, the DVD Copy Control Association anticipated that AACS would eventually be compromised. Indeed, as early as 2006, screen recording methods began to emerge as a means to bypass AACS [Sla06]. In 2007, a significant breach occurred when a key used in the encryption was leaked [Jer07; BBC07], becoming infamously known as the "illegal number" after a takedown letter sent to the search engine Google from the Advanced Access Content System Licensing Administrator (AACS LA) using the Digital Millennium Copyright Act (DMCA) [Adv07].

Both CSS and AACS represent the continuous attempts and challenges in securing physical media content. While CSS was relatively quickly broken, AACS showed some resilience before eventually being compromised.

2.1.3. Transition to Digital

The transition from physical media to digital platforms marked a pivotal moment in the evolution of DRM systems. As the digital landscape expanded to include a myriad of devices and distribution channels, DRM technologies faced new challenges and opportunities, leading to new actors emerging.

Windows Media DRM (WMDRM). Microsoft solution for securing multimedia content was released in 1999. The system incorporated a range of cryptographic algorithms, including Elliptic Curve Cryptography, DES, a custom algorithm called MultiSwap, and RC4. Despite these security measures, the system was compromised by several tools like FairUse4WM [Mar06], DRMDBG [DRM05], or Mirakagi.

Apple FairPlay. FairPlay was the proprietary DRM employed in iTunes and iPod devices. Designed by Apple in 2003, The system was built around the AES encryption algorithm to decrypt protected audio from major music companies. In 2006, Jon Johansen, one of the authors of DeCSS, claimed to have successfully reverse-engineered FairPlay [BBC06]. The following year, Apple CEO Steve Jobs published an open letter acknowledging the flaws of DRM systems [Ste07], and by 2009, FairPlay was effectively removed from iTunes.

Both these DRM systems demonstrate the ongoing battle between content protection and user accessibility. Their respective vulnerabilities and eventual modifications or discontinuations underscore the complexities involved in creating a secure yet user-friendly digital content distribution environment.

2.1.4. The Mobile World: OMA DRM

The Open Mobile Alliance (OMA), now known as OMA SpecWorks [OMA], is a nonprofit, non-governmental organization established in 2002. It focuses on developing open international technical standards for the mobile phone industry.

OMA has introduced several DRM specifications, notably OMA DRM 1.0 [Ope02] and OMA DRM 2.0 [Ope08] respectively drafted in 2002 and 2004. These DRM versions were designed to provide a secure and flexible framework for content protection across various mobile platforms, restricting, for instance, users from sharing content like ringtones and wallpapers stored on their mobile devices.

To augment content security, OMA DRM 2.0 introduced extensions known as Secure Removable Media (SRM) 1.0 and 1.1 [Ope09a; Ope09b], and Secure Content Exchange (SCE) [Ope06]. OMA SRM 1.0 and 1.1 extend OMA DRM 2.0 to secure digital rights on removable media like memory cards, with SRM 1.1 adding features like direct license provisioning. OMA SCE 1.0 enables seamless content sharing across multiple subscriber devices and improves interoperability between OMA and non-OMA DRM systems.

While still active and having published candidates for OMA DRM 2.2, OMA SpecWorks being more focused on mobile operators and networks, the mobile ecosystem has moved with the proliferation of Internet-based content, leading to modern DRM systems taking the lead.

2.2. Modern DRM Systems

As consumer behavior in digital media consumption has evolved, so too has the importance of Over-the-Top (OTT) services such as Netflix, Disney+ or PrimeVideo, prompting a transformation in the design and application of contemporary DRM systems. In recognition of this evolution, the World Wide Web Consortium (W3C) unveiled the Encrypted Media Extension (EME) [DSWB19] for HTML5, establishing it as the official web standard for DRM, influencing, in return, the evolution of DRMs.

This section presents various modern DRM systems, such as Marlin DRM, Microsoft PlayReady, Apple FairPlay Streaming, and Google Widevine. We also introduce watermarking techniques as a complementary security measure to DRM, providing a comprehensive overview of current strategies to protect digital media.

2.2.1. Marlin DRM

Initiated in 2005 by Intertrust, Panasonic, Philips, Samsung, and Sony, Marlin DRM is a DRM specification designed to accommodate a broad spectrum of digital media types and distribution platforms. This Consortium was created to bridge the limitation of the previous initiative CORAL [KCLR07], led by Hewlett-Packard and Intertrust, for DRM interoperability lacking support for different types of protected assets.

Marlin DRM was built as an open standard to unify DRM within its founders and adopters. This standard is composed and maintained by two distinct organizations, the Marlin Development Community (MDC) [Mara] and the Marlin Trust Management Organization (MTMO) [Marb]. The former, MDC, is entrusted with the task of developing and maintaining the open standard of Marlin DRM along with its associated technologies. Conversely, the MTMO functions as the licensing authority for commercial applications of Marlin DRM and oversees the provisioning of certificates and cryptographic keys.

Marlin protocols include two main contributions, the Marlin Broadband (MBB) [Mar10], and its subset the Marlin Simple Secure Streaming (MS3) [Mar17]. MBB is designed to allow persistent content protection and flexible rights management and incorporate various support for business models like subscription or rental. Its subset, the MS3, represents a lightweight version of the specification pushed by the emergence of the EME recommendations that necessitated a rapid protocol for web-based support and an authentication mechanism absent in this HTML5 extension.

The open nature of the Marlin ecosystem has not only attracted contributions from other consortiums like MovieLabs, who introduced specifications for the Enhanced Content Protection (ECP) to bring support for Secure Video Path (SVP) and Trusted Execution Environment (TEE) integration, thereby enabling the use of Marlin DRM for UHD and 4K content [Mov18], but it has also facilitated the development of SDKs for TEE support and native Android/iOS application integration by providers such as Intertrust. This comprehensive approach extends beyond audiovisual assets; the core philosophy behind the Marlin DRM specification also encompasses the protection of other media types, including e-books and byte-code [Mar14; Int22b].

Nowadays, in addition to its presence in SmartTV, user devices and Set-Top Box (STB), the Marlin DRM is used by national TV services in Italy, the United Kingdom, and France [Int22a], and is the most deployed DRM system in China, being adopted by the streaming and pay-TV company iQIYI [Int13].

2.2.2. Microsoft PlayReady

PlayReady [Mic23], initially developed in 2007 as a successor for WMDRM, has become an integral part of Windows 10 and is prevalent on Windows-based workstations and laptops. PlayReady can be integrated as an adaptable end-to-end framework into the operating system or hardware and implemented across various consumer devices, such as desktops, mobile devices, smart TVs, and STB.

PlayReady defines three security levels: SL150, SL2000, and SL3000. The SL150 level is generally used for clients under development or testing and is not suitable for commercial content. SL2000 is designed for hardened devices and applications consuming commercial content, where assets and secrets are protected through software or hardware means. SL3000 is the highest security level, aimed at devices only, and involves hardware-based protection using a TEE [Mica].

PlayReady does not provide content encryption, user authentication, or key management mechanisms for content providers, letting them be responsible for implementing these features server-side. Indeed, content providers can utilize the PlayReady Server SDK to build diverse servers compatible with PlayReady clients, encompassing content packaging, license, metering, and domain controllers. This adaptability makes PlayReady a widely used solution under a flexible model, allowing it to be implemented in various applications and devices.

In 2022, vulnerabilities in PlayReady were exploited in Canal+ VOD services, specifically due to a STB flaw and shortcomings in the server-side implementation of PlayReady [Gow22]. These incidents underscore the potential risks associated with content providers implementing server-side features, which can lead to misuse and exploitation if not correctly managed.

2.2.3. Apple FairPlay Streaming

Apple FairPlay [App23], found in Apple iOS and tvOS, operates on the FairPlay Streaming (FPS) specification. FPS is designed to facilitate the secure transfer of encryption keys to devices and enable the protected playback of encrypted content using the HTTP Live Streaming (HLS) protocol. Within the FPS framework, content providers must implement their own content server, key server, and player application. These components must comply with the FPS requirements to ensure compatibility, making it a part of the specific design structure of the system.

In addition, like Marlin DRM with byte-code protection, FairPlay serves as the cryptographic management system for the installation of iOS applications on Apple devices. Specifically, Apple employs the IPA file format for distributing applications via the Apple Store. This file format encapsulates encrypted data, which is subsequently processed by the operating system during application installation. FairPlay is responsible for securely managing the decryption key and overseeing the entire decryption process. This security measure aims to mitigate the risk of unauthorized decryption and distribution of the contents within the IPA files, thereby safeguarding potentially paid or sensitive applications from unauthorized access. Notably, FairPlay employs obfuscation techniques as an additional layer of security to deter unauthorized decryption and distribution of IPA file contents. However, these obfuscation methods have been compromised and successfully reverse-engineered, as documented in recent studies [LLM21; nic23].

2.2.4. Google Widevine

Among DRM systems, Widevine [Goo23d] is a closed-source proprietary DRM technology purchased by Google in 2010. Its early version, Widevine Classic, was compatible with older Android versions (up to Android 5.1) but only supported the *.wmv* format. Nowadays, the most recent version, Widevine Modular, or simply Widevine, supports various streaming protocols, including MPEG-DASH and CENC [ISO22; ISO16] regularly used by OTT platforms such as Netflix and Disney+. Widevine is now the most deployed DRM by being present in Android

devices, Android TVs, most web browsers, and many other devices (e.g., Chromecast, VR Headsets) [Int22a].

Widevine defines three security levels: L1, L2, and L3, where the L1 level is considered the most secure for playing High Definition (HD) videos from OTT platforms. Widevine depends on TEE to implement L1 security. At L1, both cryptography and video processing take place inside the TEE. It is worth noting that applications inside the TEE are hardware-dependent; therefore, Widevine comes with a different implementation for each. L2 and L3 are implemented where the TEE is not an option, such as legacy phones, Widevine-locked ones due to device tampering, or even desktops without a secure element. In Android, Widevine does not propose L2 security. The L3 lets cryptography and video processing occur outside the TEE. They are considered more vulnerable, given that the resulting binary, the Content Decryption Module (CDM), relies solely on software for security. As for users, L3 delivers sub-HD resolutions since Widevine recommends HD and ultra-HD content support only for L1.

Regarding its security aspect, Widevine has experienced multiple breaches over the years. In June 2016, Livshits and Mikityuk uncovered a flaw in Widevine that enabled users to access decrypted contents stored in the cache [Wir16].

Early 2019, Buchanan announced that he had successfully compromised Widevine desktop by conducting a differential fault analysis attack on its white-box implementation [Dav19]. This allowed an attacker to extract the original encryption key, thereby enabling the decryption of the media stream, an attack later repeated by Hadad on its GitHub [Had21].

In 2021, Zhao revealed the inaugural attack that compromised Widevine L1 on Android devices by successfully recovering the L1 keybox [Zha21].

While these contributions have been valuable in exposing vulnerabilities in Widevine, they have not delved into the protocol internal mechanisms and usage, which is the central focus of this thesis.

2.2.5. Watermarking Techniques

As explained in this chapter, DRM systems have evolved significantly over the years to protect copyrighted digital content. Watermarking techniques have emerged as a complementary approach to DRM in this landscape. These techniques serve as an additional layer of security that not only restricts unauthorized sharing but also aids in tracing the origin of the leak.

Two primary types of watermarking techniques are commonly used. The first is *Visual Watermarking*, where a visible watermark overlaps the video content, often a logo or text. While effective for dissuasion, this method is not ideal for tracking individual leaks. In contrast, *Forensic Watermarking* is invisible to the human eye and offers more sophistication in tracking the source of unauthorized sharing. Within forensic watermarking, there are further categorizations such as A/B Watermarking, which involves creating multiple versions of the same content to distribute uniquely constructed stream to the end user, recently specified for Adaptative Bitrate format [Lau22], and Bitstream-based Watermarking, where the watermark is embedded directly into the video bitstream.

These watermarking techniques find applications in various domains due to their widespread distribution by watermarking vendors such as NAGRA, Irdeto Synamedia, or Verimatrix [NAG; Ird; Syn; Ver]. For instance, watermarking techniques are often employed to secure live sports streams. Regarding pre-release content, watermarking is crucial for protecting precious content from piracy. Additionally, in digital cinema, watermarks applied by projectors help to identify

in-theatre camcorder-based piracy.

Watermarking has become an integral part of the DRM landscape. It offers an additional layer of security and traceability, making it a versatile tool in the fight against digital piracy. Its diverse techniques and broad application areas strengthen its role in modern DRM strategies.

3 Android DRM Ecosystem

I've seen things you people wouldn't believe. Attack ships on fire off the shoulder of Orion. I watched C-beams glitter in the dark near the Tannhauser gate. All those moments will be lost in time... like tears in rain... Time to die.

Roy Barry – Blade Runner

This chapter provides an overview of the established elements within the Android media ecosystem, focusing on media playback using the Media3 environment, ExoPlayer, and MediaDrm and MediaCrypto objects from the Android DRM framework for encrypted content.

Media3, ExoPlayer, and the Android DRM framework form the whole pipeline for secure media playback on mobile. This chapter outlines these components and their functions and references the underlying mechanisms within the Android media ecosystem. It highlights their relationships that are essential for understanding the contributions of this thesis.

Contents

3.1. Android Media2 & Media3

From a developer point of view, the Android ecosystem offers many packages to display media content to users. As of March 2023, Jetpack Media3 Application Programming Interface (API) has become the preferred method for media content delivery in Android applications [Andc]. This API was introduced to reduce the complexity and unify the media API, which had become fragmented due to the existence of various Media2 libraries serving various purposes and the redundancy offered by the ExoPlayer package. Indeed, both worlds were overlapping with Media2 user interface, media session, and player libraries acting as unused duplicated code when coupled with ExoPlayer.

ExoPlayer. At the time of writing, two versions of ExoPlayer coexist: the standalone ExoPlayer [Goo22b] and the ExoPlayer module incorporated into Media3 in 2023 [Goo23c]. The Media3 ExoPlayer serves as the standard implementation for media player interface, offering compatibility with multiple streaming protocols and default configurations for audio and video rendering, including playing protected content. The utilization of ExoPlayer minimized the need for manual implementation of basic playback functionalities. Indeed, Media3 aims to unify media API and to provide an easy approach to the Android media framework [Andb]. For developers, using Media2 or Media3 ExoPlayer for protected content allows them to abstract codec, media sampling, and media decryption when needed by leaving all Media API interconnection management to the player itself. As shown in the following section, the Android framework offers a comprehensive level of abstraction for proprietary media format and protection that ExoPlayer can use through various Hardware Abstraction Layer (HAL) modules to decrypt assets.

3.2. Android DRM Framework

In order to cope with the fragmented Digital Rights Management (DRM) ecosystem, Android offers a unified API in Java/Kotlin for DRM systems, starting from API level 18. This was implemented by a HAL module called Media DRM Server, which abstracts the actual running DRM from the programming interface used by Over-the-Top (OTT) apps [Andd] as shown in a top-down approach in Figure 3.1. At the top, applications communicate with the underlying DRM framework to instantiate and use vendor-specific implementation of DRM plugin and HAL while keeping it abstracted [Anda]. This design is the one mainly used in this thesis due to our preliminary analysis of Widevine in Chapter 5 being on an Android using such a layout. However, since Android 11, the Media DRM Server process has been replaced to direct calls to the HAL.

By zooming this overview in Figure 3.2, we can see that the Android framework offers various API for media consumption. Of particular interest for us among them are DRM API consisting of two modules: MediaDrm and MediaCrypto [Andf; Ande]. MediaDrm is used to communicate with License Servers and manage such licenses, being cryptographic keys, to decrypt a given media, while MediaCrypto performs the actual decryption. These API support the Common Encryption Scheme (CENC) for common OTT playback but also implement other encryption schemes. Each call to these modules gradually descends toward vendor-provided code by going through JNI to the `libmediadrm` responsible for communication with DRM-specific code.

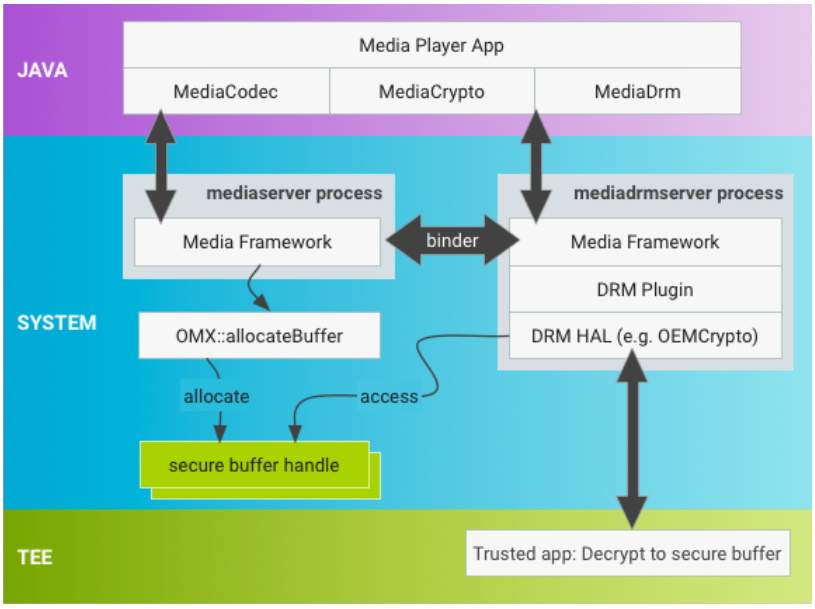

Figure 3.1. – DRM Plugin usage through the Media DRM Server (from Android 7 to 11). Figure from `https://source.android.com/docs/core/media/framework-hardening`.

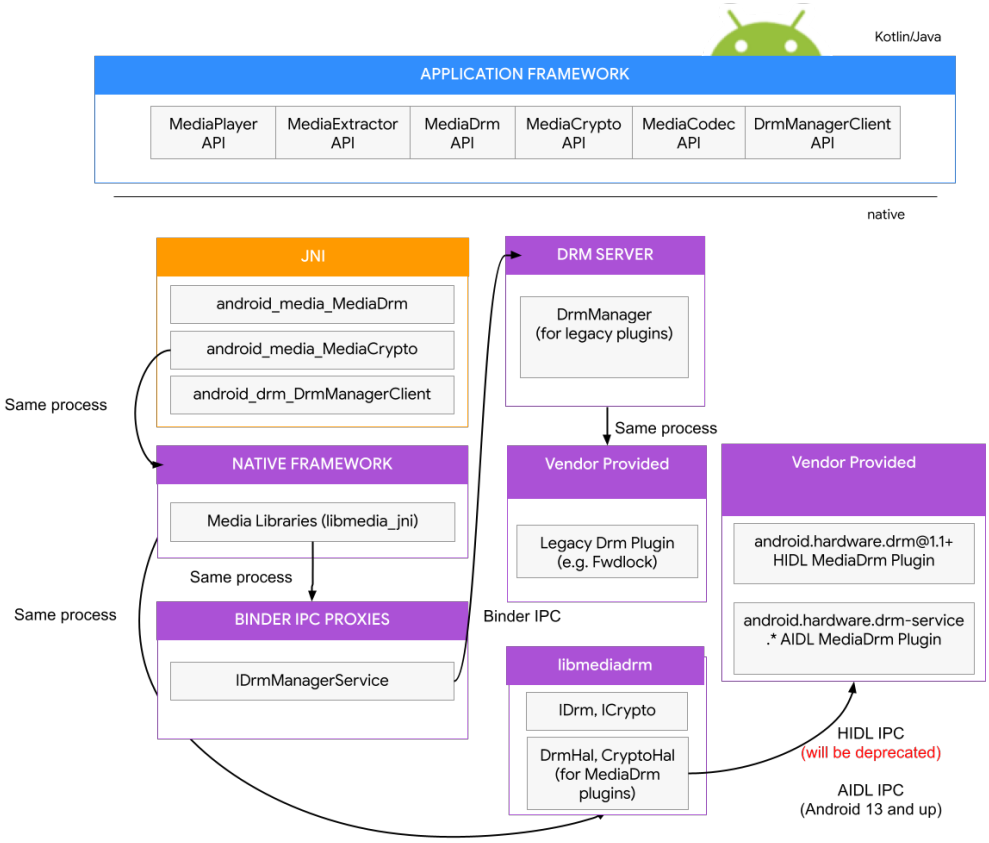

Figure 3.2. – DRM Hardware Abstraction Layer starting from Android 11. Figure from `https://source.android.com/docs/core/media/drm`.

3.2.1. Android DRM Components

This subsection focuses on the critical components of Android DRM infrastructure, specifically the `MediaDrm` and `MediaCrypto` APIs. These APIs serve as essential mechanisms for secure media playback on Android devices and are responsible for DRM license management and content decryption. The section provides a comprehensive analysis of their interactions with other Android media APIs and their role within the broader DRM ecosystem.

MediaDrm. The `MediaDrm` class functions as the central interface for decrypting DRM-secured content on Android platforms. It manages DRM licenses and offers functionality for obtaining cryptographic keys while encapsulating DRM-specific operations and communications. Additionally, it oversees the initialization and termination of the underlying Content Decryption Module (CDM), commonly referred to as the DRM plugin. The `MediaDrm` class is also responsible for creating the `MediaCrypto` object, which performs the actual media decryption.

MediaCrypto. The `MediaCrypto` class is used in conjunction with `MediaDrm` to handle the decryption of DRM-protected media during playback. It works closely with `MediaCodec`, another Android class designed for audio and video encoding and decoding. Besides facilitating the decryption of encrypted media, `MediaCrypto` also offers a secure mechanism for transmitting session keys from an operator-dedicated key server to a client device through the `CryptoSession`; a nested class within `MediaDrm`. It is used for encrypting and decrypting arbitrary data and is not limited to media content. This transmission relies on the device pre-installed Root-of-Trust (RoT) for a specific DRM plugin. It enables the execution of cryptographic operations such as encryption, decryption, signing, and verification of arbitrary user data.

Workflow for Encrypted Content Playback. Initially, the application obtains media content from a Content Delivery Network (CDN) for rendering via a media player, such as ExoPlayer. Internally, instances of `MediaExtractor` and `MediaCodec` are created. As depicted in Figure 3.3, the application retrieves the 16-byte Universal Unique Identifier (UUID) associated with the DRM scheme from the content metadata. Utilizing this UUID, a `MediaDrm` object is instantiated, compatible with the requisite CDM, which serves as the client-side DRM plugin implementation.

Following this, the application calls `openSession()` to generate a unique sessionId for subsequent interactions with the CDM. Through the `MediaDrm` object, a key request message is obtained using `getKeyRequest()`, which is then sent to a license server. The server response is fed back into the `MediaDrm` object via `provideKeyResponse()`.

Once the sessionId is acquired, a `MediaCrypto` object is created using the previously obtained UUID and sessionId. This `MediaCrypto` object is registered with `MediaCodec`, allowing the decryption of content.

After the configuration of `MediaExtractor`, `MediaCodec`, and `MediaCrypto` objects, encrypted media samples are extracted and enqueued for decryption by the CDM using the `queueSecureInputBuffer()` call.

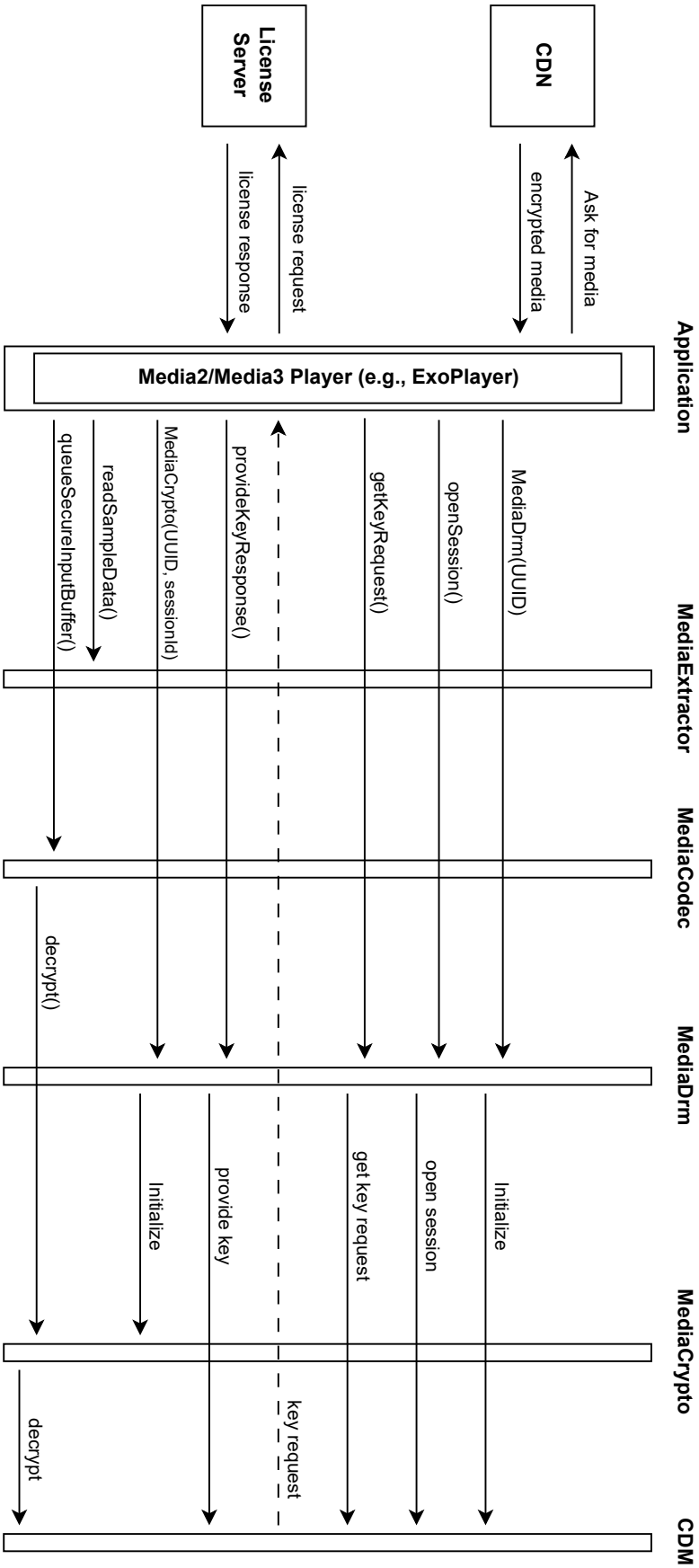

Figure 3.3. – Encrypted Content Playback in Android.

Encrypted Media Extension 4

If that's what it takes to save the world,
it's better to let that world die.

Geralt of Rivia

Due to the fragmented ecosystem of DRM solutions in the 2010s, the World Wide Web Consortium (W3C) defined the Encrypted Media Extension (EME) standard as an extension to HTML5 to provide a standardized API enabling web applications to interact with browser-supported DRMs. EME is designed to make the same web application run on any user-agent regardless of the underlying DRM implementation. Despite being optional, EME is supported by major browsers: Edge, Firefox, Chrome, Safari, Opera, and their mobile counterparts [Can23].

In this chapter, we present the different components involved in EME and detail its workflow. Then, we present privacy concerns raised in the recommendation regarding user tracking techniques.

Contents

4.1. EME Components

The EME API brings together multiple entities. Here, we list the main ones used during the control flow [DSWB19].

User-Agent. The EME implementation lies in the user-agent (e.g., a web browser) and is called through JavaScript to communicate with a CDM and distant servers using opaque messages. In the rest of this thesis, we use the terms browser and user-agent interchangeably.

CDM. The Content Decryption Module (CDM) is the client-side implementation of the DRM system in the EME specification. It is responsible for license protection and usage enforcement to decrypt media content on the user device.

CDN. The Content Delivery Network (CDN) is a distant server providing encrypted media to the user device on demand. It also offers the necessary information for license requests and DRM configurations.

License Server. License requests generated by the CDM are sent to the related license server of the given key system. This server manages licenses and policies for protected content.

4.2. Protocol Workflow

EME defines a uniform API centered on CDM session management and license acquisition while abstracting message contents. Henceforth, the EME API relies on multiple objects and events to implement the proprietary protocols of CDMs. We exclude user authentication to license and CDN servers from EME workflow due to it being out of scope of the EME specification.

License Acquisition. An overview of the protocol can be seen in Figure 4.1. For the execution, we assume that the EME user-agent has already received all the required information about the encrypted media from the CDN, such as supported codecs and used DRM system. The EME workflow starts in step 1 by the JavaScript running in the user-agent, taking as arguments the requested DRM as well as configurations for media decryption and playback. Using the self-assigned Universal Unique Identifier (UUID) [For22b] of the chosen DRM system, the user-agent asks for a handler using the `requestMediaKeySystemAccess` method of the `navigator` object 2. The EME API `createMediaKeys` is then used to instantiate the desired CDM in step 3. At this stage, the CDM is available and represented by a `MediaKeys` object but cannot receive any license. Step 4 creates a session within the CDM, represented by a `MediaKeySession` object, and identified by a session ID. Such sessions refer to key wallets and cryptographic environments isolated from each other, used to receive licenses and decrypt protected media. On creation, a session key wallet is empty and cannot decrypt anything. To receive media decryption keys, referred to as licenses, media-specific *Initialization Data* are sent to the CDM 5 to generate a license request using the `generateRequest` method. The content of this generated request fully depends on the key system in use 6 and is therefore managed as an opaque message by EME. When receiving the license request 7, the user-agent forwards it to a CDM-compatible license server 8. The opaque request is

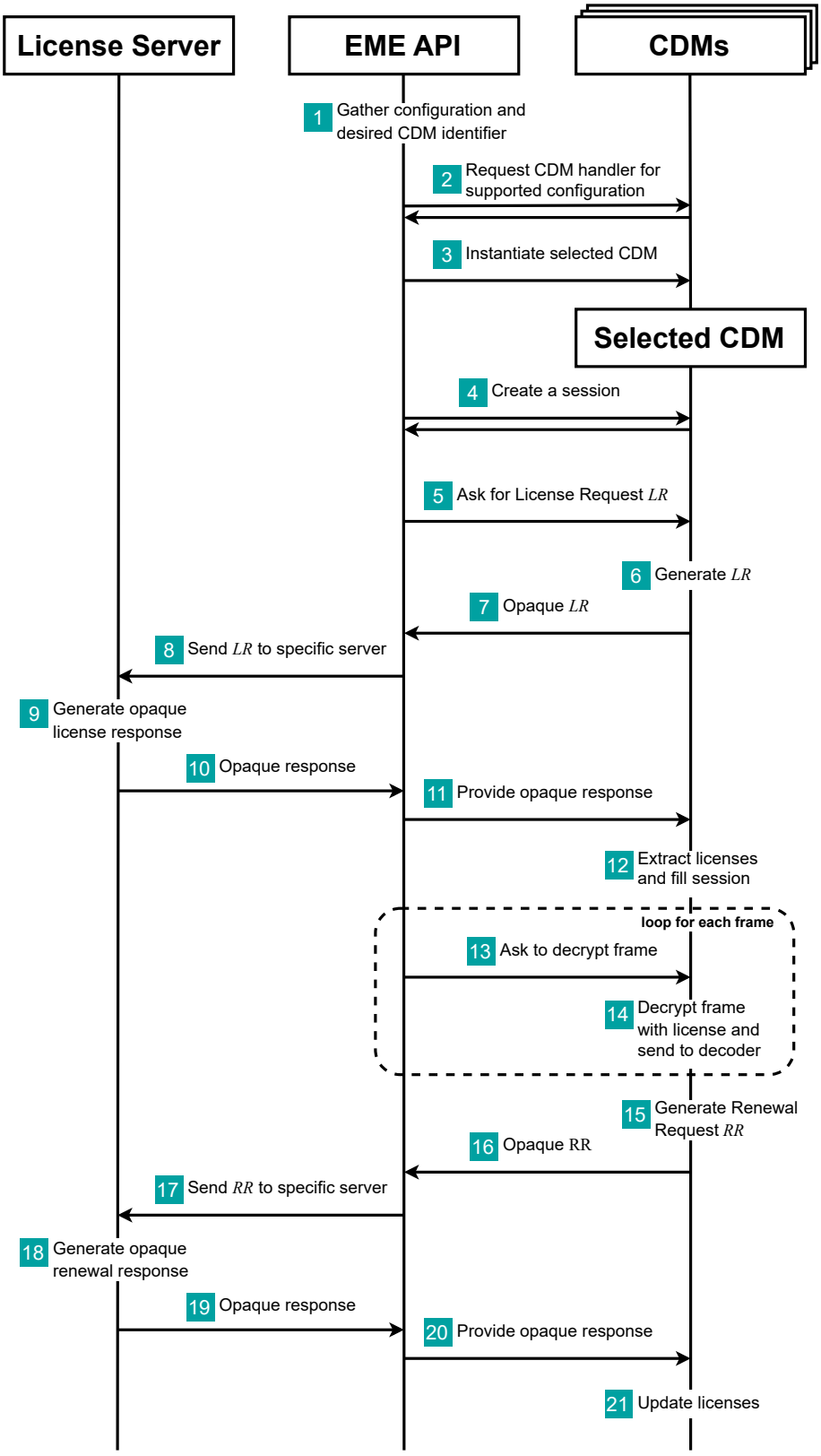

Figure 4.1. – EME Workflow: License Acquisition and Renewal.

processed by the license server generating an opaque license response in step 9 to send back to the user-agent 10. Using `update` 11, the response is sent to the CDM session and proceeded by DRM-specific mechanisms 12. Now containing at least one license, the CDM session can receive frames of the encrypted media to decrypt them 13, and forward them in clear to the decoder 14. At this point, the session can be terminated, or renewal can happen.

License Renewal. Once received, licenses can be extended by renewal mechanisms. Based on DRM-specific license policies, the CDM can be authorized to ask for key updates. After policy verification, the CDM generates a renewal request 15 and provides it to the user-agent by the `MediaKeyMessageEvent` event 16. Steps 17 to 19 are similar to the ones in the steps 8 to 10. The renewal response is sent to the CDM with the `update` method 20 to refresh session licenses 21.

Persistent Session. In addition to license request/response API, EME also defines CDM session types: '`persistent-license`' and '`temporary`' types. Standard `MediaKeySession` are temporary sessions, meaning licenses will be destroyed on session closing. For persistent sessions, the user-agent and CDM need to support offline license storage. If license policies allow a persistent state, licenses can be stored on closing for future usage (e.g., offline streaming). They can therefore be reloaded using their session ID with the `load` function of a `MediaKeySession` object to fill the newly created session in step 4 of Figure 4.1.

4.3. EME Privacy Concerns

While non-normative, the EME standard raises multiple privacy concerns, especially regarding fingerprinting, information leakage, and user tracking. The raised concerns aim to guide the DRM providers while specifying the content of the different opaque messages between license servers and CDM modules. Among these issues, the potential presence of so-called *Distinctive Identifier* and *Distinctive Permanent Identifier* is highlighted multiple times since such information could lead to user tracking. As their names suggest, these identifiers are defined as unique or shared with a small number of users that could be used to identify them. The main difference between the two is the provenance of data. Distinctive Permanent Identifiers are any distinctive value associated with or derived from data being at least non-trivial for the user to remove, reset or change, such as hardware identifier, operating system, user-agent instances, or factory values. As for Distinctive Identifiers, they represent data produced or derived from one or more Distinctive Permanent Identifiers and exposed to a component other than the CDM but with the possibility of being cleared by the OS. As mentioned in the standard, due to their characterizing nature, such identifiers could be used by malicious origins for fingerprinting on user devices and user tracking.

Within the EME standard, user tracking is further discussed in the persistent session mechanism. As explained in section 4.2, this feature allows the CDM to store licenses for future usage. The specification states that the user-agent must allow the user to clear the stored licenses as site data, such as cookies.

To limit the impact of these possible issues, EME puts forward a per-origin and per-browsing profile policy for DRM system usage and data storage. Moreover, users should be asked for consent when a CDM is needed, and this consent should be linked to a specific origin without giving away permission to untrusted websites.

Part II.

Contributions

5 Exploring Widevine Internals

Rogues are very keen in their profession, and know already much more than we can teach them.

Alfred Charles Hobbs

In this chapter, we focus on the most widespread solution, the closed-source Widevine DRM. For years, Digital Rights Management (DRM) systems have been used as the go-to solution for media content protection against piracy. With the growing consumption of content using Over-the-Top platforms, such as Netflix or Prime Video, DRMs have been deployed on numerous devices considered as potential hostile environments.

Installed on billions of devices, Widevine relies on cryptographic operations to protect content. Our work presents a study of Widevine internals on Android, mapping its distinct components and bringing out the different cryptographic keys involved in content decryption. We provide a structural view of Widevine as a protocol with its complete key ladder. Based on our insights, we develop WideXtractor, a tool based on Frida to trace Widevine function calls and intercept messages for inspection. Using this tool, we analyze Netflix usage of Widevine as a Proof-of-Concept (PoC), and raised privacy concerns on user-tracking. In addition, we leverage our knowledge to bypass the obfuscation of Android Widevine software-only version, namely L3, and recover its Root-of-Trust (RoT).

Takeaway:

This chapter supports the following contributions:

- ➤ The reverse engineering of Widevine components on Android.
- ➤ Analysis of cryptographic operations within the DRM protocol.
- ➤ Recovery of the Widevine L3 RoT, leading to full protected assest leakage.
- ➤ Design of *WideXtractor*, an automated tool to monitor Widevine inner workings and messages.

The appropriate background to grasp the details of our contribution is provided in:

- ➤ Chapter 2 to gather the context of DRM systems for content streaming.
- ➤ Chapter 3 provides the necessary information on the Android DRM framework to understand how Widevine fits within Android and vendor-specific components.

Contents

5.1. Context and Motivation

Nowadays, people prefer media consumption on Over-the-Top (OTT) platforms, such as Netflix and Amazon Prime, that distribute multimedia content over the Internet, allowing users to play them whenever they wish. Such ease of viewing the same videos across devices creates challenges for content producers and owners. The main challenge remains piracy; namely, platforms delivering media content would like to ensure that the receiving devices offer enough security at the level of hardware and software to prevent leakages. Indeed, protection by authentication is not enough, as the OTT platforms need to prevent the free redistribution of copyright-protected content.

The *de facto* answer to these challenges is Digital Rights Management. DRM is a technology that is designed to prevent piracy of digital content. It protects the content owners by restricting media consumption to authorized consumers. Despite being cried out, DRM systems were increasingly adopted on users devices. The World Wide Web Consortium (W3C) has recently published the Encrypted Media Extension (EME) [DSWB19], which is the first official Web standard for DRM, ignoring all the expressed worries [Hal17] about a web that should remain open. One of the most popular DRM solutions is Google Widevine [Goo23d], which is currently deployed on web browsers (e.g., Chrome and Firefox), Android Operating System (OS) on mobile devices and smart TVs among others. Most popular OTT players and video-streaming services, including Netflix and Disney+, leverage Widevine to protect their content.

Widevine protects video streams at several levels. At the heart of its protection is Common Encryption Scheme (CENC) [ISO15], specifying encryption standards and key mapping methods that a DRM Content Decryption Module (CDM) should implement to decrypt media files. Nevertheless, the actual key exchanges and protection mechanisms are not documented, because of the proprietary nature of Widevine.

Our Contributions Our work aims to push this topic forward, as we believe that the stakes are high regarding the Widevine protocol. This thesis intends to fill this lack of public research by providing the first thorough analysis of the Widevine protocol. Here, we overcome the restriction of signing a Non-Disclosure Agreement (NDA) to get the full description of the Widevine protocol by performing a complete reverse engineering of the Android Widevine modules. Moreover, we show that a deep understanding of Widevine can allow attackers to easily recover its internal secret parts without requiring to defeat the applied obfuscation. This is worrisome for two reasons. First, many streaming services rely on Widevine DRM to protect content against piracy. Any harm to this technology can lead to huge financial losses. For instance, in 2020, the OTT market size was estimated at $13.9 billion and expected to reach $139 billion by 2028 [For22a]. Second, zero-day vulnerabilities on Widevine can be exploited to harm countless users [Ben16] since Widevine is estimated to be installed on more than 5 billion devices around the world.

In this work, our approach was to begin with a manual analysis to gain insights into the structure of the Widevine protocol as well as its main cryptographic operations. Then, we design a Frida-based tool to automatically extract details about the Widevine workflow as it is leveraged by Android OTT apps. Afterward, we extend our tool to trace the Widevine operations within web applications running in an EME-supporting browser. Finally, we take a look at the outcome of our analysis and discuss its relevance regarding the security of the Widevine module or the OTT app. We emphasize that, throughout our work, we carefully

play the role of the security researchers who, as described by the DMCA's exemptions, act in good faith. Indeed, we timely report all our findings to Google, Widevine and Netflix. In addition, we gave up all the keys that we succeeded in extracting so that they could be revoked by the concerned parties.

Our contributions are the following:

- We reverse-engineer Widevine components on Android. In particular, we thoroughly explore its different cryptographic components from the root of trust, aka *keybox*, until the key decrypting the media, aka *Content Key*, and provide an implementation of this key ladder.[1]

- We uncover the structure of the Widevine protocol and detail the contents of the exchanged messages between Widevine and the different entities of the DRM ecosystem. We also dissect Widevine internals during the execution of an OTT app, and split them into three main operations: provisioning of device-specific RSA keys, provisioning of content license keys, and content decryption.

- In order to automate Widevine inspection, we design *WideXtractor*,[2] which is a tool based on Frida to monitor the inner workings of Widevine during media playback and dump exchange messages.

- We leverage our tool to study Netflix usage of Widevine, in order to understand how it protects its media assets: video, audio and subtitles. We find that Netflix mainly establishes two Widevine sessions: one to receive protected media, and another one for obtaining the related decryption keys. We spy on the first session using WideXtractor, and notice that Netflix, unlike other OTT apps, only protects the download URL of audio tracks, that can be effortlessly downloaded in clear even without a Netflix account. Indeed, the second session only concerns the decryption key of video tracks.

- We discuss two issues raised by our analysis. The first one is related to the use of a distinctive device identifier by Widevine. This allows third-party servers to profile users behavior during media consumption without their consent. The second one concerns a methodology that we define to efficiently recover Widevine Android software-only root of trust despite the underlying obfuscation hiding the critical parts of Widevine. We timely report all findings to Google and Netflix and were assigned the CVE-2021-0639. We were awarded by the bug bounty program of Netflix and the Vulnerability Reward Program of Google.

5.2. Widevine and Android

In this section, we describe the integration of Widevine into the Android ecosystem. In particular, we detail all the components of which Widevine consists and their interaction. This will help us to better frame our reverse engineering methodology by pointing out the relevant components to analyze in order to uncover the internals of Widevine.

[1] `https://github.com/Avalonswanderer/widevine_key_ladder`
[2] `https://github.com/Avalonswanderer/wideXtractor`

5.2.1. Widevine Components

> **Reminder:** Since Android 11, the `mediadrmserver` process has been replaced with direct calls to the HAL (see section 3.2).

> **Takeaway:** Vendor-specific code for the CDM lies in two distinct binaries: the Widevine library, software-only based L3 implementation running in the Android world, and the Widevine trustlet in the Trusted Execution Environment (TEE) providing L1 security.

In Android, Widevine comes as a dynamically loadable HAL plugin within the `mediadrmserver` process. Similar to other HAL plugins, Widevine is manufacturer-provided. In addition, it is not open-source; only provided as binary code and library files. To keep things secure, when a TEE is available, the HAL plugin delegates all sensitive operations to the Widevine component that runs inside the TEE. Roughly speaking, the resulting architecture looks like this (other components might exist depending on the Android version):

- **Widevine library.** This library is used by the `mediadrmserver` process to translate Android DRM API calls to Widevine CDM ones. The behavior of this library changes depending on the Widevine security level. In L1, it plays the role of a proxy and communicates with the TEE through `liboemcrypto.so`. As for L3, it contains the obfuscated CDM. Its name can change depending on the version and System-on-Chip (SoC) including but not limited to: `libwvdrmengine.so`, `libwvhidl.so`, `libwvm.so`, `libdrmwvmplugin.so`.

- **liboemcrypto.so.** This library performs marshaling and unmarshalling of requests to the Widevine trustlet. All communications with the TEE go through a specific TEE driver (e.g., `QSEEComAPI.so` for Qualcomm Secure Execution Environment (QSEE)).

- **Widevine trustlet.** It runs inside the TEE and implements all the needed functionalities for L1.

5.2.2. Components Interaction

Android Widevine architecture is summarized in Figure 5.1. In a top-down architecture, components interact as follows. DRM services start from the OTT application calling the Android Media Framework API to interact with the `MediaDRM` and `MediaCrypto` objects. All calls to the DRM API go through some Java Native Interface (JNI) layer via the `libmedia_jni.so` library. Calls are then forwarded to the Media DRM Server instantiated by the `mediadrmserver` process, which is the last module implemented by Android. The Media DRM Server reaches the Widevine-specific implementation through the HAL APIs. Any communication with Widevine first goes to its specific library such as `libwvdrmengine.so`. In L3, no further component is involved. As for L1, whenever CDM is required, this library calls `liboemcrypto.so` that sends the related requests to the Widevine TEE trustlet.

Of particular interest, the Widevine library does the translation between the HAL API to Widevine functions. Once translated, if Widevine is in L1 mode the Widevine API is used to call its equivalent in `liboemcrypto.so`. The OEMCrypto library role is to forge a message for the TEE trustlet containing the function arguments and command code. For a given TEE,

each Widevine function corresponds to a specific command code used by the TEE driver in order to be received by the Widevine command handler within the trustlet.

5.3. Widevine Internals

5.3.1. Methodology

Given the Widevine component layout described in section 5.2, three main components of Widevine come to light: the Widevine library, `liboemcrypto.so`, and the Widevine trustlet running in the TEE. To study Widevine inner workings, we have inspected both statically and dynamically these components starting from the Widevine trustlet.

Analysis Environment. We examine four smartphones and their respective factory images:

- **Nexus 5**: Android version 6.0.1, build hammerhead m4b30z. Widevine L3 mode with version 3.1.0.
- **Nexus 5X**: Android version 8.1.0, build bullhead bhz32c. Widevine L1 mode, version 5.1.0, security patch level 2.
- **Pixel:** Android version 10, build sailfish qp1a.191005.007.a3. Widevine L1 mode version 14.
- **Pixel 3:** Android version 11, build blueline rq3a.210805. Widevine L1 mode version 15.

All tested smartphones integrate the Qualcomm TEE (QSEE), and therefore our analysis includes some QSEE-related details. Henceforth, we will be careful to distinguish what is true for Widevine as a protocol, and what is specific to QSEE.

Trustlet Extraction. The first step to reverse the proprietary implementation of Widevine was to extract the binaries from the file system either from a physical (rooted) mobile or by downloading and extracting the phone factory images on the Google website [Goo]). These binaries, including the trustlet, can be found in the `/vendor/` directory.

For Qualcomm SoC based phones, Widevine trustlet is located in `/vendor/firmware/` and is divided into several .bXX files, where XX is a counter. To reconstruct the complete trustlet, we can simply concatenate each of these files in order to obtain a standard ELF binary with an additional hash table section used for integrity verification.

Trustlet Analysis. The Widevine L3 CDM within the Widevine library (e.g., `libwvdrmengine.so`) being obfuscated, we preferred to start with another implementation in order to figure out how Widevine works. We notice that the L1 trustlet is not obfuscated at all. Worse still, it provides verbose clear debugging strings in its internal code, thereby leading to a better understanding of the overall structure and control flow of the protocol. Leveraging tools such as *Ghidra* [Nat] and *Radare2* [Pan], we were able to retrieve function names and cryptographic keys within the Widevine protocol. Relying on our knowledge of the L1 trustlet, we then went back to the library to discern the HAL calls to L1 and to analyze the obfuscated L3 layer.

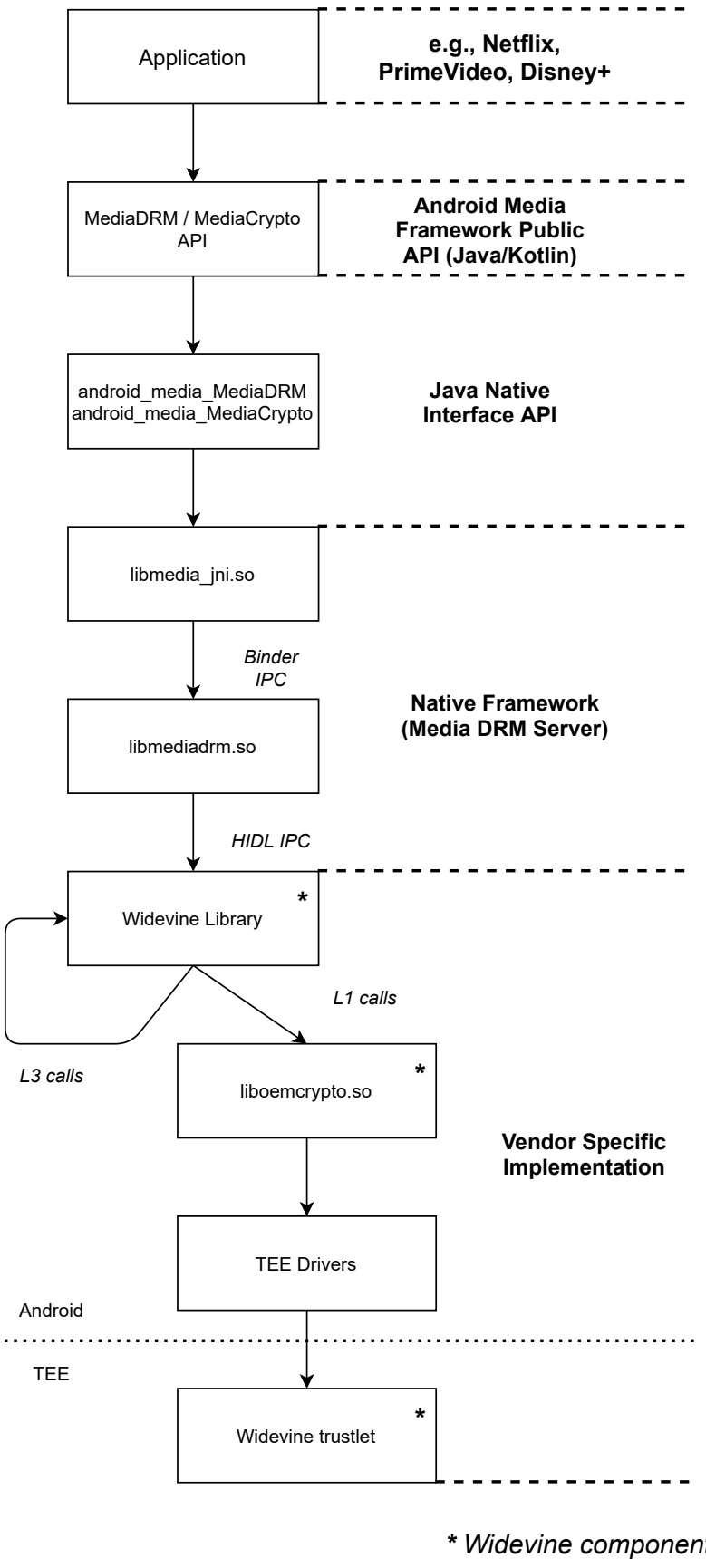

Figure 5.1. – Widevine Architecture in Android (Binder and HIDL IPC before Android 11).

Symbols	OEMCrypto Functions	Symbols	OEMCrypto Functions
oecc01	Initialize	**oecc25**	Generic_Decrypt
oecc02	Terminate	**oecc26**	Generic_Sign
oecc03	InstallKeybox	**oecc27**	Generic_Verify
oecc04	GetKeyData	**oecc28**	GetHDCPCapability
oecc05	IsKeyboxValid	**oecc29**	SupportsUsageTable
oecc06	GetRandom	**oecc30**	UpdateUsageTable
oecc07	GetDeviceID	**oecc31**	DeactivateUsageEntry
oecc08	WrapKeybox	**oecc32**	ReportUsage
oecc09	OpenSession	**oecc33**	DeleteUsageEntry
oecc10	CloseSession	**oecc34**	DeleteUsageTable
oecc11	DecryptCTR	**oecc35**	LoadKeys*
oecc12	GenerateDerivedKeys	**oecc36**	GenerateRSASignature*
oecc13	GenerateSignature	**oecc37**	GetMaxNumberOfSessions
oecc14	GenerateNonce	**oecc38**	GetNumberofOpenSessions
oecc15	LoadKeys*	**oecc39**	isAntiRollbackHwPresent
oecc16	RefreshKeys	**oecc40**	CopyBuffer
oecc17	SelectKey*	**oecc41**	QueryKeyControl
oecc18	RewrapDeviceRSAKey	**oecc42**	LoadTestKeybox
oecc19	LoadDeviceRSAKey	**oecc43**	ForceDeleteUsageEntry
oecc20	GenerateRSASignature*	**oecc44**	GetHDCPCapability
oecc21	DeriveKeysFromSessionKey	**oecc45**	LoadTestRSAKey
oecc22	APIVersion	**oecc46**	Security_Patch_Level
oecc23	GetSecurityLevel	**oecc47**	LoadKeys*
oecc24	Generic_Encrypt	**oecc48**	DecryptCENC

* Duplicated entries differ in version.

Table 5.1. – OEM Crypto Library Symbols Equivalents

Libraries Analysis. Our reverse engineering of the trustlet allowed us to map the Widevine protocol functions called OEMCrypto with function symbols (`_oeccXX` for L1 and `_lccXX` for L3). The correspondence for Widevine Modular L1 functions is summarized in Table 5.1. We also leveraged the *Frida* toolkit [Rav] to trace the execution of these functions in the `mediadrmserver` process, as both the Widevine library and `liboemcrypto.so` are instantiated in the Media DRM Server. This leads us to figure out the workflow of the Widevine protocol regarding the called operations as well as the related cryptographic keys.

5.3.2. Widevine Protocol

Takeaway: The Widevine protocol comes with three main steps:

- **Certificate Provisioning** allows the device to receive an intermediate RSA RoT for license acquisition.
- **License Provisioning** represents the request/response message exchange between the CDM and a license server to recover a content key for media decryption.
- **Content Decryption** uses the received content key to decrypt protected assets.

In section 5.2, we present the components enabling protected content playback within Android devices using Widevine. Taking a look at the bigger picture to highlight the communications behind these elements, we can distinguish 7 agents: a Content Delivery Network (CDN), a provisioning server, a license server, an OTT application, the Android Media servers, the Widevine library and the Widevine CDM.

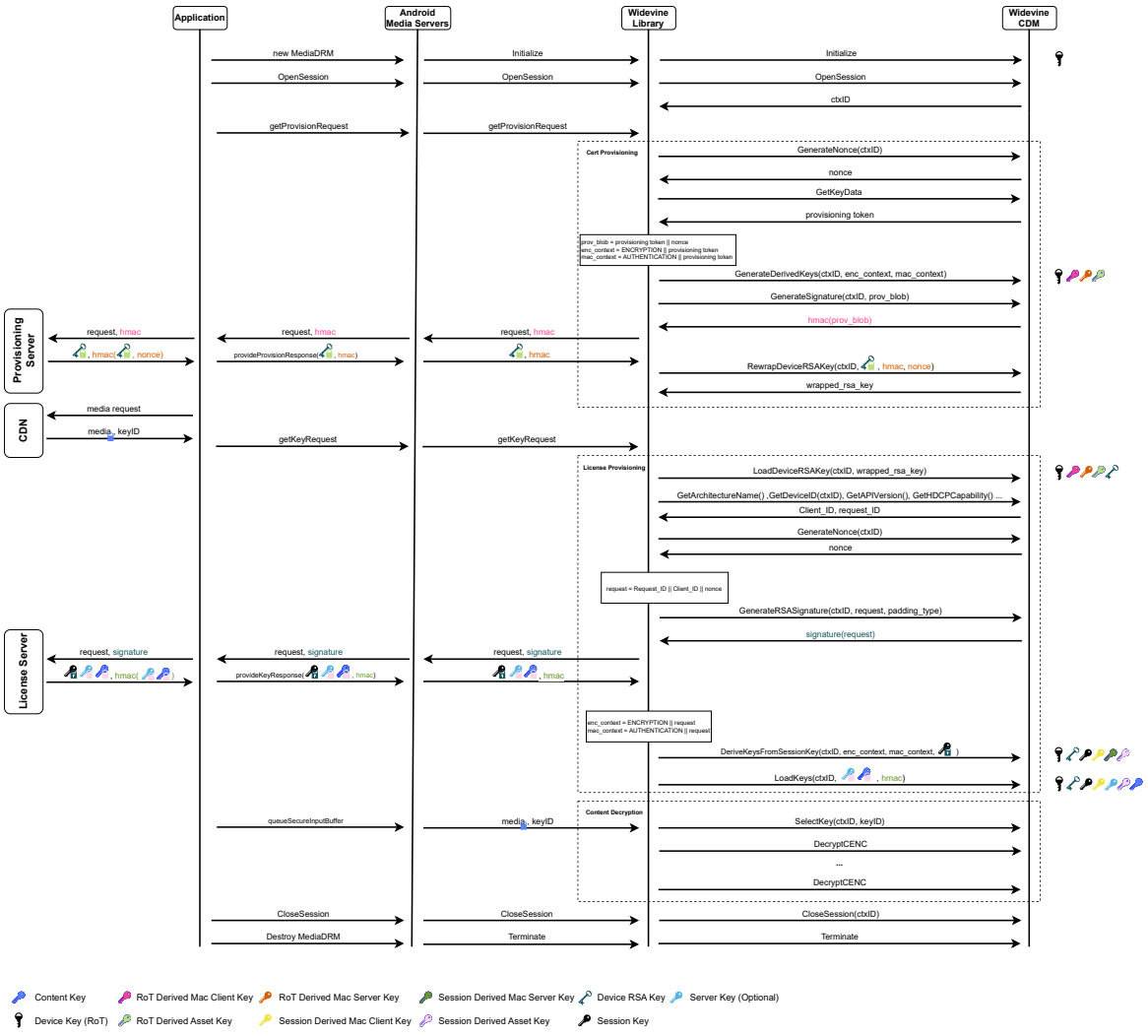

Figure 5.2. – Widevine Protocol for Media Decryption

Indeed, when playing protected content using an OTT app (e.g., Netflix), content decryption is managed by the Android Media Servers that rely on the underlying DRM system, here Widevine. All key requests for provision and license servers are generated by Widevine components, especially the library, with the help of the CDM. Overall, the Widevine protocol involving these actors is divided into three main phases: Certificate Provisioning, License Provisioning and Content Decryption. An illustration of the protocol can be seen in Figure 5.2.

Certificate Provisioning. The provisioning phase is usually done once to recover a cryptographic certificate and does not need to be done for future media decryption. The private key within this certificate protects the fresh session keys. A new request is sent to the provisioning server when no certificate can be found, the one installed is corrupted, or the OTT needs to install a new certificate.

On request creation, the CDM generates a nonce to ensure freshness. Then, it derives keys for certificate decryption and integrity checks, based on the Widevine RoT called the *Device Key*, and dynamically generated buffers. These buffers are based on a token within the RoT structure detailed later. Using `OEMCrypto_GenerateSignature`, the request is HMAC-protected with the RoT derived client key and sent to the provisioning server.

The received response is passed to the CDM through the `OEMCrypto_RewrapDeviceRSAKey` function. After nonce check and integrity verification using the RoT derived server key, the certificate is decrypted using the previously derived key and stored on the persistent storage of the device after being rewrapped (i.e. re-encrypted) by a device-specific key. This marks the end of the installation process of the certificate private key called the *Device RSA Key*.

License Provisioning. The OTT receives all required information about the protected content from the CDN in order to ask for the corresponding *Content Keys*, also known as license keys. After loading the stored certificate using `OEMCrypto_LoadDeviceRSAKey`, a request is forged using a generated request ID and various device-specific info concatenated within a device blob called the Client ID. This message is then signed by the Device RSA Key and sent with a newly generated nonce to the License Server.

The corresponding response will be received by the CDM through `OEMCrypto_DeriveKeyFrom-SessionKey` that will extract and decrypt a *Session Key* using the *Device RSA Key*. The *Session Key* is then used to derive other HMAC and encryption keys based on buffers containing a dynamically generated Client ID containing the *Device RSA Key* public key, more details on the Client ID are given in chapter 7. These keys are later used to verify the integrity of the response received by `OEMCrypto_LoadKeys` and to decrypt the *Content Keys*. Here, Content Keys are associated with a Key Control Block (KCB), encrypted or not by its related content key. KCB contains the previous nonce and various information that we will detail in subsection 5.4.3. It is important to note that, once all Content Keys have been added to the CDM, the nonce is cleared from memory. During key reception, the License Server can provide a new Server Key protected by the derived asset key. This key constitutes a new server MAC key for integrity verification of future responses, such as in `OEMCrypto_RefreshKeys`.

Content Decryption. Note that more than one Content Key can be loaded in the CDM memory at the same time. Therefore, the right one for the media is selected using its key ID in `OEMCrypto_SelectKey` before content decryption within `OEMCrypto_DecryptCENC`.

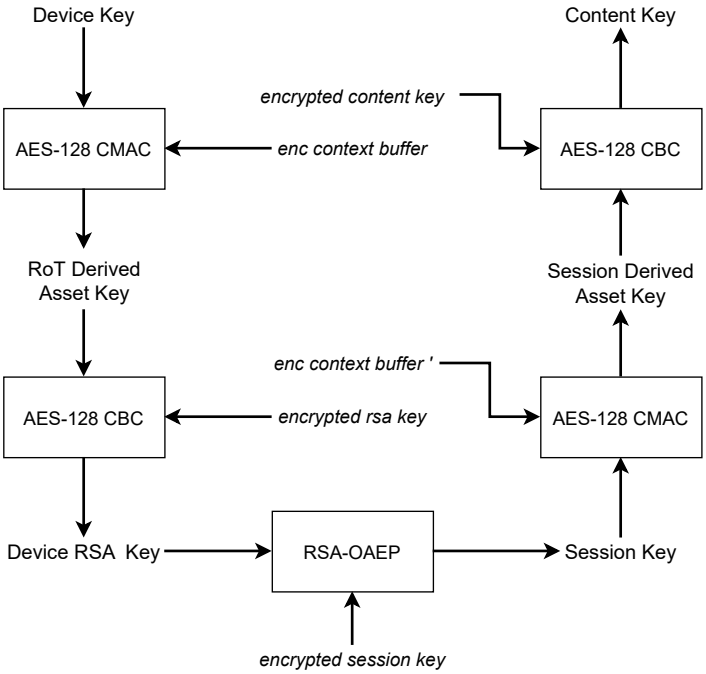

Figure 5.3. – Widevine Key Ladder

5.4. Widevine Cryptography

> **Takeaway:** Widevine Android devices come with a unique factory-provisioned RoT called the *Device Key*. This AES 128-bit key can be used during the certificate provisioning phase to obtain a *Device RSA Key* used for *Session Key* decryption and *Content Key* acquisition.

In this section, following our reverse engineering analyses, we succeed in depicting a complete picture of Widevine cryptographic mechanisms. In particular, we uncover the Widevine internal key ladder from its root of trust to the content decryption key. An overview is summarized in Figure 5.3. Note that our study holds for both L1 and L3. Moreover, we provide an implementation of the key ladder in Python.[3] For ethical and legal purposes, we did not include the root keys. Thus, nobody can use our project to actually pirate OTT content.

5.4.1. Widevine Root of Trust: Keybox

Widevine, in [Wid], mentions that its RoT is established by a factory-provisioned component called the *keybox*. While doing our static analysis, we noticed that this keybox is mainly used to secure the provisioning of certificates. We will discuss more about this in subsection 5.4.2. Now, let us focus on the nature of this RoT and where it is stored.

Keybox storage. While exploring the structure of the Widevine trustlet, we looked at the Qualcomm Secure File System (SFS) that allows trustlets to encrypt their sensitive data before

[3] `https://github.com/Avalonswanderer/widevine_key_ladder`

Field	Description	Size (bits)
Device ID	Internal Device ID	256
Device Key	128-bit AES key	128
Provisioning Token	Used by provision requests	576
Magic Number	"kbox"	32
CRC32	CRC32 validating the keybox integrity	32
Total		**1024**

Table 5.2. – Widevine KeyBox

storage. On our setup, the SFS persistently stores the encrypted files on `/persist/data/sfs/`. Then, we continued our exploration and found the function `init_tzdrm_config_path` initializing paths for several elements. Here, the string names are explicit, and we think that the *keybox_ lvl1.dat* file refers to the keybox for L1. We also found the `OEMCrypto_InstallKeybox` function that can be called from `libwvdrmengine.so` to re-install a factory keybox supposed to exist in the `/factory/wv.keys` file. Nevertheless, we have never found such a file. As for L3, we noticed that the library `libwvdrmengine.so` loads the keybox from the `ay64.dat` file that can be found in `/data/mediadrm/IDMXXXX/L3/` or `/data/vendor/mediadrm/IDMXXXX/L3/`.

Keybox structure. In order to figure out of what the keybox consists, we looked deeper at the trustlet functions that verify keybox integrity, because they display an explicit error log message that clearly identifies the concerned field. We began with `OEMCrypto_isKeyBoxValid`. Indeed, this function shows us that the keybox structure is 128 bytes long with two special fields at the end. The trustlet checks the integrity of the decrypted keybox by verifying that the last eight bytes are composed of a magic number *"kbox"* followed by a 4-byte Cyclic Redundancy Check (CRC) code. In this work, we will not discuss the effectiveness of this integrity verification despite being broken in other contexts [GGK+16]. We retrieved the remaining fields by looking at the functions loading the keybox: `OEMCrypto_GetKeyData`, `OEMCrypto_GetDeviceID`, and the internal function `OEMCrypto_GetDeviceKey`. The *Device ID* is a 32-byte unique device identifier for Widevine. The *Key Data*, or the *Provisioning Token* as referred to by `libwvdrmengine.so`, is 72 bytes long and is used within provisioning requests. The remaining 16 bytes correspond to an Advanced Encryption Standard (AES) key that is called, according to the API, *Device Key*. This key is the real RoT. We summarize the five fields of the keybox in Table 5.2.

5.4.2. Device RSA Key

As explained in section 5.3, Widevine does not directly use its RoT to protect licenses. Instead, it leverages an RSA key pair that, unlike the keybox, can be installed dynamically on the device through a process called certificate provisioning. This process is protected by the *Device Key* that is derived into several keys in `OEMCrypto_GenerateDerivedKeys`. Upon reception, the `OEMCrypto_RewrapDeviceRSAKey` function first verifies the integrity of the key pair by re-computing a 256-bit HMAC tag. Then, it decrypts the key pair and re-encrypts it again with a device-unique (TEE-specific in L1 or keybox-related in L3) 128-bit AES key. These keys never leave the CDM. This function also checks the key pair format after decryption and before re-encryption. Indeed, the key pair, aka the *Device RSA Key*, is

expected to be an RSA certificate with PKCS#5 padding in PKCS#8 format as indicated by the `qsee_secpkcs8_parse` and `get_len_with_pkcs5_padding` functions.

Once re-encrypted, aka re-wrapped, the *Device RSA Key* is stored on the standard file system by the Widevine library in a `cert.bin` file on `/data/mediadrm/IDMXXXX/`. Widevine distinguishes L1 from L3 by using different directories. This persistent data is later used in future instances of the Widevine CDM to avoid making new provisioning. Here, `OEMCrypto_LoadDeviceRSAKey` is used to recover the stored certificate. We note that re-wrap is MAC-and-Encrypt, as it also computes a 256-bit HMAC tag on the key pair. Please note that a new provisioning process is performed whenever the `cert.bin` file is deleted or corrupted.

5.4.3. Content Keys

Keys protection. As introduced previously, content keys, or license keys, are obtained from `provideKeyResponse`. Here, Widevine first calls `DeriveKeysFromSessionKey` from `OEMCrypto` to decrypt a special field; *Session Key*, using the *Device RSA key*. This key is later used to derive a 128-bit AES key as well as two 256-bit HMAC keys. Then, these keys are used in `OEMCrypto_LoadKeys` to decrypt the license key and verify its integrity.

Key Control Block. Alongside the Session and Content Keys, the response from the License Server also contains additional 128-bit metadata called KCB, one for each license key in the response, and is encrypted by its associated *Content Key*. The KCB is identified by the magic number *kctl* or one of the forms *kcXX*, where XX is related to Widevine version. The *Content Key* is accepted only when the associated KCB is checked by `verifyKeyControlBlock`. This function helped us to understand the structure of KCB: a nonce, Time-To-Live (TTL) of the key, and 32-bit of control bits. These control bits define usage rights (e.g., encryption, MAC tag generation, etc.) and anti-rollback features. During the lifetime of a *Content Key*, the KCB can be updated through the `OEMCrypto_RefreshKey` function that, unlike its name might indicate, cannot change the key or usage rights but only its TTL. Such refresh requests work as license ones with the exception that the `OEMCrypto` function `GenerateSignature` is used for integrity protection instead of `GenerateRSASignature`.

Keys Usage. During loading in the CDM memory, Content Keys are stored in a key table structure with an identifying key ID for `OEMCrypto_SelectKey`. For media playback, encrypted buffers are decrypted with the chosen key by `OEMCrypto_DecryptCENC` implementing MPEG-CENC.

5.4.4. Nonces

The Widevine protocol mitigates replay attacks and ensures message freshness using nonces both present within requests and KCB. By `OEMCrypto_GenerateNonce`, the Widevine library can ask the CDM to generate up to 20 nonces per second stored in a First In, First Out (FIFO) queue of 16 elements within the CDM memory. These 32-bit nonces are generated using a Pseudo Random Number Generator (PRNG) and used at each request creation and response reception. If the nonce is valid, it is removed from the queue and the message is processed, otherwise the message is dropped. During *Content Keys* loading in `OEMCrypto_LoadKeys`, a single nonce can be used in multiple KCBs. In this case, the nonce is only removed once all keys have been processed.

5.4.5. Summary of Cryptographic Algorithms

> **Takeaway:** The Widevine CDM offers various cryptographic operations such as AES 128-bit encrypt, decrypt, HMAC-SHA256 sign and verification. And provide RSA encryption and signatures such as RSA-OAEP-SHA1, RSASSA-PSS-SHA1 and RSASSA-PKCS1-v1_5.

Widevine Generic Crypto API. In addition to media decryption, the Widevine CDM allows applications to perform arbitrary cryptographic operations within a dedicated session. In Android, using the `CryptoSession` class of Media DRM, an application can leverage the underlying DRM plugin to protect data using the `OEMCrypto_Generic_XXX` family of `Encrypt`, `Decrypt Sign` and `Verify` functions. Here, each operation must have the appropriate key usage rights.

Key Derivation. Widevine never uses received or stored keys directly. Instead, it relies on key derivation algorithms, implemented in `OEMCrypto_GenerateDerivedKeys` and `OEMCrypto_DerivedKeyFromSessionKey`, in order to generate three different keys each time: a 128-bit Asset key, a 256-bit MAC Client Key, and a 256-bit MAC Server Key. The leveraged PRF (Pseudo-Random Function) is AES-128-CMAC to generate the required 640 bits. In addition to secret keys, the derivation algorithm uses two buffers, called *encryption context* and *mac context*, that are created based on device-unique information and used respectively for the Asset key and both MAC keys. For each chunk, the related license request is appended to a string that begins with a string counter and the word "*ENCRYPTION*" for encryption context or "*AUTHENTICATION*" for the mac ones. Only one encryption context with counter "1" is needed, while the mac context includes four counters starting from 1.

Symmetric Cryptography. All CDM operations related to key decryption, generic encryption API (i.e., `Generic_Encrypt`, `Generic_Decrypt`), and media protection are performed using AES 128 bits. Both key decryption (e.g., `OEMCrypto_LoadKeys`) and encryption API implement AES in Cipher Block Chaining (CBC) mode, while media decryption relies on `OEMCrypto_DecryptCENC` which supports MPEG-CENC (i.e., AES-128-CTR).

AES Initialization Vectors. Within key decryption functions, IVs are being received alongside their respective cyphertext in server responses. On *Device RSA Key* rewrap, Initialization Vector (IV) generation is handled by a PRNG algorithm with `OEMCrypto_GetRandom`. For protected content, MPEG-CENC standard is used to deal with IVs.

MAC Generation. The MAC Client Keys and Server Keys respectively protect requests and responses to provisioning and license servers using HMAC-SHA256. The same algorithm is used by the `OEMCrypto` API `Generic_Sign` and `Generic_Verify` using the currently loaded Content Key.

RSA Operations. The *Device RSA Key* includes a 2048-bit private key that is used for both decryption and signature during *Session Key* loading and license request creation. For decryption, this key is used in the RSA-OAEP-SHA1 mode, while RSASSA-PSS-SHA1 and RSASSA-PKCS1-v1_5 can both be used for requests depending on the function argument.

5.5. WideXtractor

Most OTT apps, including Netflix, Disney+ and Hulu, apply anti-debugging mechanisms in order to prevent attackers from easily intercepting and tracing calls to Widevine. In addition, our reverse engineering shows that it is quite demanding to untangle the Widevine interface between its different components. This is due to the fact that dissecting the Widevine workflow provides an important insight into its internals. Thus, we implement WideXtractor; a tool tracing the Widevine calls made by an OTT app. In this section, we present an overview of the design and implementation of this tool. Then, we show the effectiveness of our tool by highlighting interesting findings while analyzing the most downloaded OTT app on Android, namely Netflix. Finally, we extend WideXtractor to inspect Widevine as a plugin on Chrome.

5.5.1. WideXtractor Design

> **Reminder:** Widevine messages in Android all go through its software-only `_oeccXX` functions and then dispatch either to the L3 or L1 implementations (see section 5.3).

We design WideXtractor to automatically trace the execution flow of Widevine on Android. Our goal is twofold. First, researchers can easily and systematically study how OTT apps leverage Widevine while displaying protected content. Interesting findings can be revealed by analyzing the actual calls and their parameters, as we demonstrate for Netflix in the following subsection. Second, our insights about the used operations make the tool uncover the secret keys that should exist in the memory at a particular point in time, despite the applied obfuscation.

We implement WideXtractor using Frida to monitor all calls to the `OEMCrypto` functions in the Media DRM Server. Monitoring the `mediadrmserver` process has two advantages: (1) it allows us to bypass anti-debug techniques at the application level, such as SafetyNet [And22], and (2) both L1 and L3 workflow can be recovered.

Our monitoring traces any call to Widevine functions, hence the underlying protocol outline, while dumping the used arguments, such as buffers for requests and opaque reply data. To this end, WideXtractor relies on a Frida server running on the Android device with *sys_ptrace* capability. This can be achieved by running as the media group or a system/root privileged user. Our script hooks the `OEMCrypto` functions after attaching to the `mediadrmserver` process instantiating DRM libraries.

Once launched and attached, WideXtractor logs every method of the Widevine protocol and receives buffers from the Android OS. Our traces correspond to the library symbols that we reverse-engineered and summarized in Table 5.1. Our tool allows attackers to inspect exchanged messages during key reception and media decryption before opaque requests and responses. WideXtractor can be found on our Github.[4]

5.5.2. Case Study: Netflix Message Security Layer

> **Reminder:** The Android MediaCrypto object offers to application a way to use underlying CDM as cryptographic oracles for arbitrary data (see section 3.2).

[4] `https://github.com/Avalonswanderer/wideXtractor`

Takeaway: By using WideXtractor, we were able to spot the Message Security Layer (MSL) protocol defined by Netflix. Using the Widevine CDM as an oracle, Netflix transmits additional data such as content download URLs without relying solely on HTTPS security.

Leveraging WideXtractor, we automatically monitor the use of Widevine by the Netflix app, which is the most popular OTT with 200 million subscribers around the world [The22]. Our analysis shows a large number of calls to the Widevine Generic Crypto API compared to other OTT apps. This is due to Netflix using a custom protocol named MSL [Net20] to protect its communication and extend it to Android using Widevine.

Following our observation, we dig deeper to understand Netflix internals. We find that Netflix requires to manage two Widevine sessions: one to get *Content Keys* and decrypt protected media, and one to exchange data using the Widevine Generic Crypto API. Henceforth, we will call these two sessions *License Session* and *Generic Crypto Session* respectively. Both sessions are initialized in the same way until `OEMCrypto_LoadKeys`.

Indeed, the License Session loads the *Content Key* that decrypts the displayed media, while the Generic Crypto Session loads several keys for confidentiality and integrity protection of arbitrary data. We note that *Content Key* can only be accessed through the `queueSecureInputBuffer` method from the `MediaCodec` class. Thus, only keys loaded within the Generic Crypto Session can be used to decrypt arbitrary data using the Android `CryptoSession`. Here, each key has its own usage rights to perform specific cryptographic operations. As explained in subsection 5.4.5, Widevine associates these functions to the following `OEMCrypto` ones: `Generic_Encrypt` and `Generic_Decrypt` for AES operations, and `Generic_Sign` with `Generic_Verify` for HMAC tag computation and verification.

Our study shows that all communications with the Netflix CDN go through the Generic Crypto Session. Thus, Netflix avoids relying solely on HTTPS to protect assets. For instance, from the Android OS view, the app asks the Widevine CDM to decrypt and verify the received messages. The decrypted data are sent back to the app without any particular protection. Therefore, by monitoring `OEMCrypto_Generic_Decrypt`, we were able to intercept all exchanged messages between the CDN and the Netflix app. These messages contain download URLs for `timedtexttracks` (for subtitles), `video_tracks` and `audio_tracks`. Each of these categories contains multiple links corresponding to different languages for subtitles and audio in addition to different qualities for video. Although the downloaded videos are encrypted using the Widevine CDM, both audio and subtitles can be obtained in clear. We perform several experiments with the audio and subtitles URLs to evaluate their validity. We find that they are accessible from any platform (PC web browsers, smartphones, tablets), by anyone (no need for Netflix account), from anywhere (no location verification), and for a limited period of time (12 hours approximately). We also find that there is no limit to simultaneous accessed devices.

Our learned lesson is that Netflix seemingly makes it harder to spy on messages sent by the CDN by adding an extra layer of protection provided by Widevine. Thus, attackers might be clueless, since defeating Android certificate pinning is not enough. However, WideXtractor easily allowed us to identify the use of the Widevine Generic Crypto API. Thus, it becomes even more straightforward to obtain the exchanged messages in clear by just recovering the returned buffer of `OEMCrypto_Generic_Decrypt`. The advantage of our approach is that we no longer need to bypass certificate pinning implemented by the OTT app. We were surprised that Netflix does not protect audio tracks with a Content Key. During our responsible

disclosure, we discovered that Netflix was not even aware of that, because they believed that non-Dash mode was sufficient. We went further and analyzed six other popular OTT apps: Disney+, Amazon Prime Video, Hulu, HBO Max, Starz and Showtime. We find that, unlike Netflix, all of them encrypt their audio tracks with the Content Key, more details can be found in chapter 6.

5.5.3. WideXTractor over EME

Similar to the unified DRM API of Android, the World Wide Web Consortium (W3C) defines the Encrypted Media Extensions (EME) standard to provide a standardized API enabling web applications to interact with the browser-supported DRM. EME is designed to make the same web application run on any browser regardless of the DRM implementation. Despite being optional, EME is supported in major browsers: Edge, Firefox, Chrome, Safari, Opera, and their mobile versions [Can23].

The logic of the EME standard is quite similar to the Android DRM system. Indeed, when the web application attempts to play an encrypted video, it starts by creating a `MediaKeys`, which is the object providing access to the CDM. Then, it calls `createSession` to instantiate `MediaKeySession` managing the lifetime of a DRM license. Next, the `MediaKeySession` object generates a license request by calling `generateRequest`. This message is sent to the license server to require the necessary decryption keys. Once the response is received, `MediaKeySession` calls the `update` method to parse the obtained license inside the CDM. Now, we can decrypt the media using the keys loaded from the license.

In PC browsers, Widevine comes as a plugin in different browsers, such as Firefox and Chrome, supporting the EME standard. Indeed, our work focuses mainly on Android Widevine. In order to overcome this limitation, we study the Widevine flow as it is implemented within the browsers providing EME. Here, we note that the CDM software is obfuscated and hides its symbols. Therefore, we follow a different approach: instead of hooking the browser EME functions, we implemented a browser plugin that intercepts all EME-related data. Then, we parse these data and compare them with the ones obtained in Android Widevine. We notice a big reciprocity between the Widevine messages in Android and PC browsers. This confirms that the Widevine protocol in Figure 5.2 works similarly in different systems. The main difference that we noticed is that the Widevine RoT in the browser consists of a white box implementation of the *Device RSA Key*.

Thus, we extend WideXtractor to trace the Widevine flow by merely looking into the EME-received messages. Our approach has the advantage of successfully following the Widevine flow without regard to the applied obfuscation or the actual called functions. Based on a Chrome EME logger plugin [Goo22a], as in WideXtractor we log buffer values and use key usage info from `update` calls to identify the message purpose within the Widevine protocol. This allowed us to log additional information to link EME calls to Widevine functions.

5.6. Security Evaluation

Widevine enthusiastically pitches the virtue of its DRM solution. Widevine being proprietary, there is no easy way to verify the security claims of this piece of software running on billions of devices. The goal of our reverse-engineering efforts is to go beyond this market irrationality. In this section, we show how our study conveniently helps in highlighting a gap between what

Widevine promises and its technical solution. The raised issues concern not only OTT but also final users.

5.6.1. Privacy Concerns

The Widevine protocol comes with privacy concerns for users in the streaming ecosystem. These issues are due to the need of Widevine to identify users devices for bailing purposes. Indeed, Widevine collects device-specific data, and sends them to distant servers, such as the provisioning or license ones. For instance, in Android these data includes the Widevine Device ID within the Widevine keybox, and the Client ID containing several device-identifying fields, such as the device architecture, phone model, CDM version, or build info.

Ironically, Widevine commits to respecting users privacy. As a matter of fact, Widevine claims to follow the EME standard. Despite being non-normative, user-tracking issues are being pointed out in the privacy section of the EME standard [Can23]. However, the usage of *Distinctive Identifier* or *Distinctive Permanent Identifier* allows origins crossing information to spot a single user based on these device-unique values. This is harmful to privacy since it allows third-party servers to profile users behavior during media consumption. Moreover, users never consent to such device tracking. A complete analysis of such issues is detailed in chapter 7.

5.6.2. Recovering Widevine L3 RoT

Widevine presents their DRM for OTT platforms as a solution to protect them from piracy. There exist several levels of compromise; each one relates to some cryptographic keys in the key ladder. Obviously, RoT recovery constitutes the most severe compromise level, since attackers can derive all keys allowing them to decrypt any protected content. Widevine distinguishes L1 RoT and L3 RoT, as it is more challenging to compromise L1 compared to L3. Indeed, Widevine relies on software-only protection mechanisms to hide L3 RoT. It is true that such protection is brittle and doomed to be broken. However, advanced obfuscation techniques might make the compromise quite involving and resources demanding. Here, we show how our understanding of the Widevine protocol may allow attackers to get L3 RoT without specific knowledge of the underlying obfuscation in an automated matter.

As explained in subsection 5.4.1, the Widevine RoT is encapsulated inside a keybox that is used to initiate the key ladder in order to retrieve clear content. Starting with certificate provisioning, the RoT is also used in L3 to protect the received *Device RSA Key* for persistent storage (i.e. rewrap operation) or using keybox-related data in the Client ID. Accordingly, we build the following approach to recover L3 RoT. We know that, by design, the RoT must somehow be loaded during the execution of the Widevine protocol, but the applied obfuscation hides the loaded RoT. Here, we rely on WideXtractor to better discern the moment, where the RoT is actually in the memory in clear. At this point, we dynamically analyze all memory regions used during obfuscated cryptographic operations within the Widevine library. We search for the keybox structure (e.g., magic number, device ID). Thus, we were able to recover the L3 keybox on a Nexus 5, including the 128-bit AES *Device Key*, due to an insecure storage of sensitive information (CWE-922). Our method is efficient since we limit the spatial and temporal memory monitoring.

Being the root of trust, we are motivated to recover the keybox. Widevine maintains a different keybox for the different levels of security. In subsection 5.4.1, we explained that

L1 protection is TEE-dependent. In QSEE, it is based on the Secure File System, whose security is outside the scope of this thesis. Here, we will focus on L3 keybox. Note that L3 implementations are diverse. Our analysis shows that Widevine is as secure as the weakest one, since license keys for a given media are shared among all L3 implementations. Therefore, someone might take advantage of outdated implementations to break into Widevine. Indeed, they can intentionally display content on vulnerable smartphones, so that they can easily recover protected media. This works as long as OTT platforms keep support for old Android smartphones, as they target a wide audience. In this chapter, we study the L3 of Google Nexus 5 which still runs many OTT apps.

By taking a closer look at `libwvdrmengine.so`, we notice that `OEMCrypto` L3 functions are obfuscated. This makes our analysis more complex, since the keybox is only used within these functions. Moreover, we find that all obfuscated functions apply anti-reverse transformations, such as control flow flattening, that make static analysis less relevant. In addition, memory regions are mapped with read and execute permissions. Because of ARM architecture blurring line between code and data, we find it hard to tell if these mapped regions are destined for data to load or code to execute.

The approach that we followed to recover the keybox was not to directly break into the layer of obfuscation. This would have made of our work technology-dependent, while we aim for more long-term lessons. Instead, we stepped back and monitored the unprotected functions calling the `OEMCrypto` interface using WideXtractor. Indeed, we notice that most functions of `libwvdrmengine.so` are not protected. Thus, we managed to collect a lot of memory data loaded during the execution of the obfuscated functions. Of particular interest, we were able to observe all memory unmapping that happens through calls to `munmap`. We noticed that the `OEMCrypto` functions load a significant amount of data through these calls especially sensitive ones.

Thus, our next target is to look for a function requiring the keybox for its operations. We recall that the keybox is regularly used in `OEMCrypto_LoadDeviceRSAKey` to decrypt the rewrapped Device RSA Key in L3 mode, but also during Client ID creation with `OEMCrypto` methods like `GetDeviceID` or `GetKeyData`. Accordingly, these functions map a proper region of memory for the keybox, load the keybox value inside it, and finally unmap that region at the end of the function. Because of the obfuscation, it is hard to observe the loading step. However, these functions do not clear the memory before unmapping. Therefore, we retrieved the content of the unmapped regions. Then, relying on what we know about the keybox, we filtered this content to keep the regions of size 128 bytes including the keybox magic number.

It turns out that there is only one. We verify our finding by checking the CRC-32 value. The keybox being recovered, we can now decrypt the license keys, hence the video tracks destined for L3. It is worth noting that this is particularly interesting, especially since we did not even get to break into the underlying obfuscation. In fact, our analyses were guided by the conceptual structure of the Widevine protocol.

5.7. Responsible Disclosure

Our findings have been timely reported to all concerned parties following their responsible disclosure process. Netflix was quite responsive and we got rewarded via their bug bounty program. Regarding Google Widevine, our security report was assigned with the highest priority within the Google Vulnerability Reward Program (VRP). The Widevine security

team investigated our findings and issued a patch to mitigate our identified flaws. Google assigned the CVE-2021-0639 for us, and acknowledged us in the Google Hall of Fame and the Android Security Acknowledgments. Our goal is to improve the knowledge about DRM, and not to provide copyright infringement tools.

WideLeak: OTT Assets and Widevine in the Wild

6

That, detective, is the right question.

Dr. Lanning – I, Robot

Becoming a major platform for streaming, Android provides its own Digital Rights Management (DRM) framework. Thus, Over-the-Top (OTT) platforms need to adapt their apps to suit Android design, despite a fragmented ecosystem and little public documentation about the underlying DRM systems. Unfortunately, the security implications of how OTT apps leverage Widevine, the most popular Android DRM, have not been studied yet.

In this chapter, we report the first experimental study on the state of Widevine use in the wild. Our study explores OTT compliance with Widevine guidelines regarding asset protection and legacy phone support. With the evaluation of premium OTT apps, our experiments bring to light that most apps adopt weak and potentially vulnerable practices.

Takeaway:

This chapter supports the following contributions:

- ➤ First empirical study on OTT apps leverage of Widevine in Android.
- ➤ Real-world impact due to misunderstanding of DRM weaknesses exposing OTTs to huge asset leakage.

The appropriate background to grasp the details of our contribution is provided in:

- ➤ Chapter 2 is instrumental in understanding the context of DRM systems as they apply to content streaming.
- ➤ Chapter 3 supplies the essential knowledge about the Android DRM framework, allowing for an understanding of how Widevine correlates with Android.

Contents

6.1. Context and Motivation

The world has almost fully moved over the earlier days of owning media. Instead, many people watch videos on over-the-top platforms (OTT), such as Netflix and Amazon Prime, that distribute multimedia content over the Internet. Today, the most prevalent model is subscription-based services, where media can be played as often as the user wishes, but becomes unavailable when a user stops paying for the service. The large distribution of media in what can sometimes be hostile environments, like untrusted devices, creates challenges for content producers. The main challenge remains piracy; in 2020, the OTT market size was estimated at $13.9 billion and expected to reach $139 billion by 2028 [For22a].

Thus, content providers rely on DRM, which is a technology that aims to protect media from illegal distribution. Modern DRMs ship content in an encrypted form, and then control their decryption through authorized players on users devices. DRM systems are often cried out because of their insecurity. However, the recent surge of OTT platforms has led to the ongoing adoption and evolution of such systems. For instance, World Wide Web Consortium (W3C) has published, despite being criticized [Hal17], the Encrypted Media Extension (EME) [DSWB19], as the first official Web standard for DRM. Android devices have become a major platform for streaming content consumption. This has drawn interest from OTT platforms to develop mobile apps in order to reach a large audience in a thriving business ecosystem. Android has anticipated such a trend and provided a DRM framework allowing the decryption of protected media streams. This framework supports many DRM systems; which DRM a device supports varies regarding the device manufacturer. The most dominant one is Widevine [Goo23d], which is also deployed on web browsers (eg., Chrome and Firefox), and smart TVs among others.

The ecosystem of DRM in smartphones is quite peculiar. First of all, starting from Android 7, Widevine takes advantage of hardware-backed security, such as TrustZone-based Trusted Execution Environment (TEE) [SAB15]. In the past decade, much work has been done to evaluate the security of software-only DRM solutions based on obfuscation and cryptographic whiteboxes [WSKV13; CWW+18]. TEE-based Widevine arguably provides stronger protection, despite some successful attacks in the literature [Zha21; Ben16; CSFP20; MGS+17]. Second, the Android DRM API, is quite different from the standardized EME specifications that are defined for the Web. Therefore, app developers need to rewrite a large part of their code, which may lead to inconsistency in security practices. This is especially true given that Widevine public documentation is sparse, which limits community discussions and sharing. Third, any DRM system has a long history of hacking and patching. Thus, they require to be updated on a regular basis. Unfortunately, it is not common for smartphones to receive long-term support from their manufacturers. Even Google used to only guarantee three-year support for their Pixels [Goo22c], before Pixel 6.

In order to address such concerns, Widevine has issued a set of guidelines and recommendations to follow by OTT apps. However, little has been done to explore how these apps implement Widevine recommendations to protect their content. In this chapter, we focus on whether the security features offered by Widevine are properly leveraged. For instance, we empirically study if apps actually encrypt their contents with Widevine keys. One might argue about the relevance of such a question, especially since OTT developers are well aware of piracy threats. However, our evaluation shows that, for example, Netflix, which is the leading app with more than one billion installations, does not even encrypt audio tracks and just delivers them in clear. Our goal is to identify missing conformance of OTT apps

regarding three Widevine security guidelines: content protection, usage of encryption keys, and support of discontinued devices.

We performed our evaluation on premium OTT apps. One might expect that Android has defined its DRM API, while taking into account Widevine guidelines since both are owned by Google. A surprising result of our work is to uncover that there is some gap between actual implementation practices and what Widevine wants. By switching to TEE-based by default, the hope was that content piracy would become old news. Nevertheless, we show that OTT apps might introduce new attack vectors rendering Widevine protection ineffective. We demonstrate the practical impact of our dire observations by obtaining DRM-free content from six popular OTT apps, including Netflix, Hulu and Showtime. We insist that we do not look to introduce yet another attack against DRM systems but to better frame the misunderstood pitfalls of the rapidly growing Android DRM. We assess their prevalence, showing a rather concerning scenario.

Our contributions can be summarized as follows:

1. *New findings and understandings.* We conducted the first empirical study on how OTT apps leverage Widevine in Android. Notably, we automate app monitoring by intercepting all calls to the Widevine process related to the DRM API. Our study reveals that almost no OTT app follows the Widevine recommendations in terms of cryptographic operations and revocation policy.

2. *Real-world impact.* To confirm the potential threats of our findings, we shed light on how certain practices allow an easy long-term compromise. We show that old still supported Widevine protection can be easily broken by using their software-only root of trust recovered in chapter 5, allowing clear access to OTT assets.

3. Our automated monitoring tool and Proof-of-Concept source code are available online[1] for reproducibility purposes and to assist future work on this topic.

6.2. Research Questions

Despite the recent surge of streaming on Android devices, little has been done to understand how Android most popular DRM, namely Widevine, is used in practice within the Android DRM framework. Our work takes the first step in this direction and explores the implementation choices of OTT apps regarding Widevine security guidelines. To better frame the scope of our study, we also quantify the usage of Widevine in the wild.

The goals of DRM solutions are purely economic: they are designed to enable the business models of content providers so that users can have access to a large catalog of movies as long as they pay their subscription. Once they unsubscribe, they cannot watch their movies again, since they actually never owned them. This model creates incentive gaps between subscribers and OTT providers with respect to the security of DRM systems. Indeed, users care less about applying security updates to their DRM systems, while OTT providers would like these systems to keep improving their security.

By using embedded Android DRM solutions like Widevine, OTT services rely on smartphone updates to provide security patches. Therefore, one can argue that the protection of their

[1] `https://github.com/Avalonswanderer/widevinel3_Android_PoC`

media assets entirely depends on the underlying implementation abstracted by the Android DRM API. Nevertheless, depending on the choices made by OTT, security can also be linked to asset delivery and license distribution. By offering various levels of security to OTT and its own license acquisition mechanism, Widevine and in particular its usage by OTT needs to be investigated to apprehend security concerns.

Our evaluation study starts from a simple observation: pirated content keeps appearing online for illegal download. Intuitively, this can be caused by the brittle security of DRM solutions. We argue that there might be another reason. Indeed, complex design and little public documentation can lead app developers to make wrong choices in their DRM adoption. Thus, we investigate the following research question:

How do OTT apps follow Widevine recommendations to protect content?

By addressing this question, we show OTT apps would introduce new attack vectors that render DRM protection ineffective while leveraging Widevine. Accordingly, we aim to answer the following four questions, described below.

6.2.1. Q1: How often do OTT apps rely on Widevine?

The main benefit of the unified Android DRM API is that developers can effortlessly rely on Widevine, or any Android-integrated DRM, to protect content. The first question we intend to answer is how effective this API is for Widevine adoption. In particular, we intend to quantify the OTT apps leveraging Widevine or custom DRM implementation like in the Indian music industry [DKS21]. Because most apps do not disclose whether Widevine is supported (an official list can be found in [Goo23d]), we need to fill this gap by conducting several experiments for both L1 and L3. The support for L1 is an important security trait since it resists most attacks against software-based DRM.

6.2.2. Q2: Do OTT apps encrypt their media contents using Widevine?

In any DRM ecosystem, Content Delivery Network (CDN) servers deliver media content to OTT apps. In a standard workflow, these contents are encrypted, and DRM modules in users devices (aka CDM) are required to decrypt them first. Nevertheless, the prevailing way of media consumption makes this operation less straightforward. Indeed, in most OTT apps, users are allowed to change the language (i.e., audio and subtitles) at any moment without negatively impacting users experience. Thus, video tracks are obtained only once, while subtitles or audio are downloaded again for each language selection. In practice, these parts are sent separately and thus might be differently protected depending on the OTT app that might choose for instance to only protect the video and not the audio. Moreover, the Android documentation of DRM API does not clearly explain this common scenario, which might inadvertently lead to unprotected content. Our work investigates whether all media assets are DRM-protected for OTT apps: video, audio and subtitle tracks.

6.2.3. Q3: Do OTT apps use multiple keys for content encryption?

Widevine has issued a set of recommendations concerning DRM cryptographic key usage [Goo22g] (similar ones are defined by W3C DRM standard EME [Can23] adopted by Widevine [Goo22f]). These recommendations state that videos with different quality

settings should be encrypted using a different key and that audio files should be protected by an additional key. Obviously, such guidelines are defined to minimize the impact of a content key recovery and therefore should be enforced. However, the use of multiple keys might make the code to manage more complex if no external media playback library is used, such as ExoPlayer [Goo22b]. Consequently, there might be some tension between Widevine recommendations about using multiple keys and app implementations. In this work, we intend to quantify the DRM keys received for each media. If an app does not encrypt audio tracks or does not use distinct keys, we note such a practice as minimal.

6.2.4. Q4: Do OTT apps still support L3 outdated devices?

Widevine regularly updates its software. For instance, the release cadence for Android CDM is annual (per Android release). In order to keep away some vulnerable implementations, Widevine may revoke devices due to non-compliance with their security rules, e.g. no longer receiving security updates. Nevertheless, OTT apps can ignore such revocation and deliver content keys to users [Goo22g]. This practice is not recommended, since old Widevine implementations could be broken at some point without any security patch. The problem of outdated devices is not straightforward, because following revocation rules restricts the number of OTT potential clients. Here, we identify to which extent OTT apps still support potentially vulnerable Android devices to underline the delicate trade-off between availability and security. We emphasize that this problem is quite different from the fact that Widevine, by design, supports L3 that is inherently bypassable [Git20]. Indeed, breaking into legacy L3 should be easier than the latest L3 version. In addition, it should have a longer impact, because no vulnerability will be patched.

6.3. Experimental Evaluation

As stated previously, our goal is to analyze the implementation choices of OTT apps while leveraging Widevine. In this section, we first describe our experiment setup to address our research questions, and review the methodology that we used to perform our evaluation. Then, we present our findings. Finally, we show how some of the existing practices can be abused to bypass DRM by targeting its weakest link.

6.3.1. Evaluated Apps

In our work, we evaluate the following Android OTT apps (with their number of installations in millions at the time of writing): Netflix (1,000M+), Disney+ (100M+), Amazon Prime Video (100M+), Hulu (50M+), HBO Max (10M+), Starz (10M+), myCANAL (10M+), Showtime (5M+), OCS (1M+), and Salto (1M+).

The selection was based on the list of Widevine clients [Goo23d] as well as their popularity on the Google Play Store. Unfortunately, we could not extend our study to more apps because platforms restrict their client from registering with a bank account in a particular country. However, our code can be easily applied to other apps.

6.3.2. Methodology

We could have answered all our research questions by looking directly into OTT apps. However, we did not consider such an option for two main reasons. First, they are closed-source, and their design and implementation differ considerably. This would have required a substantial amount of reverse engineering that is specific to each app. Second, most evaluated OTT apps apply anti-debugging techniques such as time checking [XZL+21], or rely on SafetyNet [And22] to hinder any dynamic analysis. Therefore, we preferred to focus on the Widevine HAL plugin. In particular, we precisely identify the workflow for each app, by monitoring all Widevine activities involved in the DRM API. This approach presents multiple advantages. Indeed, it bypasses all protection mechanisms implemented by apps to block dynamic analysis. Moreover, it can be easily automated. Nevertheless, in order to address some of our questions, we will still need to note OTT-specific information. Accordingly, we design our methodology considering a human in the loop, which also implies limiting the number of evaluated apps. Furthermore, the lack of documentation on Widevine made the manual analysis necessary before implementing our monitoring tool.

> **Reminder:** In Android before version 11, the `mediadrmserver` process hosts the Widevine software-only implementation acting as the dispatcher between secure and nonsecure functions. By monitoring these, we can observe message content and Widevine usage for all running apps (see section 5.5).

DRM API Monitoring. Here, we take a two-pronged methodology. First, we decompile the Java classes of the evaluated OTT apps to identify some of the included Android classes. More specifically, we scan all calls to `MediaDrm` and `MediaCrypto` methods that are required within a Widevine session, as explained in Chapter 3. However, we are aware that some apps might include some dead code. Thus, in order to err on the side of soundness (i.e., low false positives), we monitored Widevine component functions linked to the Android Media DRM framework while playing protected content. Indeed, for both L1 and L3 security levels, we intercept and note any function called within the CDM process linked to the Widevine protocol (namely `_oeccXX` functions), loaded in `mediadrmserver` starting from Android 7 and `mediaserver` otherwise. To this end, we leverage Frida [Rav] to hook CDM calls. The use of L1 is confirmed whenever the control flow reaches `liboemcrypto.so`, since L3 is characterized by the fact that all calls stay within `libwvdrmengine.so` (the Widevine library name can differ between devices). In our Github[2], we provide a Frida script that automates OTT app monitoring. Moreover, to allow more in-depth analysis, we dumped input and output buffers related to various functions, including non DASH mode (e.g., generic encrypt and decrypt).

Content Protection. By protection, we mean cryptographic encryption by Widevine. When an OTT app asks for some media, the related CDN server does not deliver it directly. Instead, it sends several URI links that are used by apps to download all required video, audio and subtitle tracks. Contents might come encrypted or clear. Our approach mainly consists of obtaining these URI links for each OTT app, to determine whether the downloaded contents are protected or not. For this purpose, we just rely on video or audio players to read the downloaded files. As for subtitles, we check whether they contain ascii characters for English ones.

[2] `https://github.com/Avalonswanderer/widevinel3_Android_PoC`

OTT	Widevine used (Q1)	Content Protection (Q2) Video	Audio	Subtitles	Widevine Key Usage (Q3)	Playback on L3 discontinued phones (Q4)
Netflix	●	Encrypted	Clear	Clear	Minimum	●
Disney+	●	Encrypted	Encrypted	Clear	Minimum	◐
Amazon Prime Video	●†	Encrypted	Encrypted	Clear	Recommended	●†
Hulu	●	Encrypted	Encrypted	-	-	●
HBO Max	●	Encrypted	Encrypted	Clear	-	◐
Starz	●	Encrypted	Encrypted	-	Minimum	◐
myCanal	●	Encrypted	Clear	Clear	Minimum	●
Showtime	●	Encrypted	Encrypted	Clear	Minimum	●
OCS	●	Encrypted	Encrypted	Clear	Minimum	●
Salto	●	Encrypted	Clear	Clear	Minimum	●

◐ Widevine fails during provisioning phase.
† using custom DRM if only Widevine L3 is available.
Minimum: Audio in clear or using the same encryption key as the video.
Recommended: Audio and Video are encrypted with different keys.

Table 6.1. – Widevine Usage and Asset protections by OTTs

The main challenge in our approach is that apps also protect these URI links, and thus we have to find them. First, we keep monitoring all function calls to the CDM process, and more particularly non DASH mode, as it might be used as a secure channel to protect arbitrary data. In addition, we intercept the received network packets when selecting and displaying protected content to recover media URI and Media Presentation Description (MPD) files. Because of the use of TLS, we require to implement SSL repinning technique using Burp [Por22] and Frida. Moreover, we note the used key IDs for each content by parsing the MPD files and their related OTT-specific metadata. Finally, we perform our experiments for L1 and L3 to assess that it does not depend on security level.

Outdated Device. Our approach is straightforward: we use Nexus 5 phone to display content. Released in 2013, it did not receive an official Android 7.0 in 2016, and thus Android 6.0.1 becomes its last update. It runs Widevine in L3 mode with CDM version 3.1.0. This is quite old, as the current CDM version is 15.0 in 2021. In our experiments, we attempt to display some media content on Nexus 5. We also keep monitoring all calls to Widevine. We distinguish two cases: (1) the app can display Widevine protected content, and (2) the app uses Widevine, but no content can be displayed.

6.3.3. Results

> **Takeaway:** While relying on the Widevine Content Decryption Module (CDM) for asset protections, most OTT apps do not comply with Widevine guidelines regarding multiple keys usage and device revocation.

Our results are summarized in Table 6.1.

Q1. All the evaluated apps depend on Widevine for content playback, and not on some custom app-specific DRM solution. One exception is Amazon Prime Video using an embedded Widevine library when just the L3 software-only mode is available. We also find that the L1 TEE-based mode is popular, unlike what was predicted in [CWW+18]. This is an important trend since TEE becomes mandatory starting from Android 7.

Q2. Based on our findings about Widevine usage, we check whether OTT apps protect their contents. We first look for URI links related to content assets: video, audio and subtitles. First, using public Frida resources, we succeeded in bypassing SSL repinning on all OTT apps, which shows how ineffective such a security mechanism is. Then, we analyzed the network packets for each app. Once we found the URI links for a given media, we downloaded the associated assets and checked their protection status. A notable exception is Netflix which creates a secure channel between Widevine and the CDN to protect URIs through the non-DASH Widevine API. This protection does not prevent us from recovering Netflix links by intercepting the output of some Widevine functions. Overall, we obtained all URI links for all OTT apps, except the subtitles URI for Hulu and Starz. Our investigation finds that video tracks are always encrypted, contrary to subtitles that are always in clear (we could not confirm this for Hulu and Starz). The case of audio tracks is more peculiar. Most OTT apps encrypt their audio, except for Netflix, myCanal and Salto. In fact, for these apps, audio in any language can be played anywhere without any OTT account.

Q3. As noted in Q2, most OTT apps protect their video and audio. Here, we explore whether they are encrypted with different keys. Of course, we cannot tell by just looking into the encrypted files. Thus, as explained previously, we analyzed some metadata indicating the identifier for every decryption keys. Our observations show that all evaluated OTT apps properly encrypt their videos with different keys depending on the resolution. This implies that we cannot trivially recover HD resolution, while breaking L3 mode. However, given the same resolution and the same content, many OTT apps are confirmed to use the same key for both video and audio, which is not recommended for DRM solutions by Widevine or EME. As for apps that do not protect audio, namely Netflix, myCanal and Salto, Widevine also considers this practice as a minimal recommendation setting. Only Amazon Prime Video strictly follows Widevine recommendations by always using distinct keys. For Hulu and HBO Max, we were unfortunately not able to conclude our analyses due to some regional restrictions.

Q4. On our Nexus 5 with its Android 6, the installation process succeeds for all OTT apps. Connected to a valid OTT account, we attempted to display some media. We notice that Disney+, HBO Max and Starz reject to provision Widevine with valid licenses, implying that Nexus 5 is actually revoked. Thus, media contents and their decryption keys were not even delivered to our Nexus 5. However, the remaining seven apps do provision Nexus 5 as a valid device and therefore display the required media. One restriction was on the video quality since Nexus 5 only supports L3, hence no HD video.

Insights and Learned Lessons. Our evaluation highlights the large adoption of Widevine by OTT apps on Android devices. More importantly, we find that TEE-backed Widevine is widely leveraged thanks to the unified Android DRM API. This shows that OTT apps are aware of piracy threats, and therefore look to rely on modern security technologies, such as TEE. However, we also find that this does not prevent developers from opting for bad security practices. Indeed, OTT apps do not protect subtitles. There might be two reasons for this. First, OTT frameworks do not consider subtitles as a valuable asset to protect. Second, there is no API to deal with encrypted subtitles in the Android DRM API. This might also uncover the rationale behind why most apps do not follow Widevine recommendation to use distinct keys for video and audio. The case of subtitles is more special though. Indeed, we notice that many apps call DRM API through ExoPlayer as recommended by Widevine [Goo22f].

This playback library proposes some API allowing developers to provide encrypted audio and video, but not subtitles. However, this API is not yet widely used with different keys. We highlight an unexpected result in our study: Netflix does not protect audio tracks. During our responsible disclosure, we discovered that Netflix was not even aware of that, because they believed that non-Dash mode was sufficient. App developers should not take all the blame, since the Android API does not make it easy to implement Widevine guidelines. However, we think that our results are surprising, since Widevine recommends ExoPlayer providing a demo app with best practices [Goo22b].

Finally, we shed light on the problem of license distribution. We notice that many OTT apps care more about reaching more clients than applying the revocation rules of Widevine: they allow the playback of protected content on devices no longer receiving security updates. Note that most Android devices have a short security update support period with for instance three years for Google Pixels [Goo22c] and five only for the latest model. This might seem anecdotal, but we will show in subsection 6.3.4 how an attacker might exploit a weak Widevine implementation to recover media contents.

6.3.4. Practical Impact

We argue that discontinued phones constitute one of the weakest links in the DRM ecosystem. Indeed, the threat model of DRM includes attackers that have full control of their devices. This is due to the fact that the same media, with different resolutions, is played everywhere, and the only incentive of an attacker is to recover that media, regardless of the targeted device. We demonstrate the practical impact of our results by obtaining DRM-free content from all OTT apps still supporting old devices (except Amazon).

> **Reminder:** By reverse engineering the Widevine internal (chapter 5), we found three main keys:
>
> - **Keybox**: 128-byte structure including a magic number and a 128-bit AES *Device Key*. This key is installed by the manufacturer and constitutes the Root-of-Trust (RoT).
> - **Device RSA Key**: 2048-bit private RSA key. This key is installed when needed by the Provisioning Server. The installation process is protected by the keybox. This RSA key protects the *Content Key* sent by the License Server.
> - **Content Key**: 128-bit AES key for media decryption.

During our reverse engineering. we find that the keybox is used to initiate the key ladder. By dynamically monitoring memory regions that are used during obfuscated cryptographic operations within `libwvdrmengine.so`, we searched for specific keybox structure (e.g., magic number). Thus, we succeeded in recovering the L3 keybox on our deprecated Nexus 5 as explained in subsection 5.6.2.

Once we recovered the keybox, we were able to obtain the provisioned *Device RSA Key*. Then, we mimic the rest of the key ladder by intercepting Widevine function arguments to recover derivation buffers and encrypted keys. We implement this key ladder to automatically recover the media-related *Content Key*. During our experiments, we found out that OTT apps use the same keys for all their subscribers for a given media. Finally, we use MPEG-CENC [ISO15] to decrypt all protected contents. With some processing, we reconstruct the

pirated media and play it on another device (i.e., personal computer) without any OTT account. Since keys depend on the required quality and all media in our experiments was requested from Widevine L3, the best quality that we get is unsurprisingly 960x540, which is qHD (quarter high definition).

Our Proof-of-Concept (PoC) of the key ladder can be found in our GitHub[3]. Following our ethical approach, we only provide the implementation of the key ladder without the associated *Device RSA Key*, nor keybox. Thus, nobody can use it to actually pirate OTT contents. Note that our PoC works for both L1 and L3. However, we only applied it on Widevine L3. Our purpose is not to define yet another attack against DRM systems, but to show that some outdated L3 security protection can lead to a simple long-term compromise because of discontinued devices. We stress the fact that there is currently no clear solutions to the upgradeability of deprecated phones, even if the situation might be improved in the future. For this reason, we believe that OTT apps must strictly abide to Widevide revocation rules to avoid piracy.

6.4. Discussion

6.4.1. Not Just Another Attack

We insist that our work does not present a new attack against DRM solutions. Instead, we show that bad implementation practices imply new attack vectors that can be abused to recover DRM-free content in Android smartphones. In particular, we show that OTT apps do not use Widevine protection to the fullest, and this might be due to a lack of public documentation (therefore community discussion) and complex proprietary design. Moreover, we show that DRM protection is quite brittle if we include all the supported, yet outdated, versions. We successfully demonstrate that deprecated phones create more risks than benefits. Note that no SafetyNet or anti-screen recording techniques can be of any use, since attackers only need to monitor Widevine which runs in a different process.

6.4.2. Limitation & Future Work

As any study, our work is not exempt from limitations. Here the focus was on the most popular DRM solution on Android smartphones, namely Widevine. However, Huawei devices offer their custom DRM solution, called WisePlay. Moreover, outside the smartphone ecosystem, Widevine is present on Android TVs, web browsers on different operating systems, and small devices (e.g., Chromecast). Studying similarities and differences among these different implementations constitutes the main future direction of this work. We hope that our tools will encourage more work on these topics to reach other platforms and CDMs with fully automated approaches.

Another limitation is that our PoC can only recover video with sub-HD quality. It is not clear how Widevine enforces the control on video transmission regarding the security level. On PCs, the GitHub project `netflix-1080p` [tru22] explains how to get HD quality on L3 by just modifying the profiles to be sent to the CDN. This implies that there is no strong verification for web browsers. An interesting future work is to adapt this exploit to Android in order to get the license keys of HD content without breaking into the Widevine L1.

[3] `https://github.com/Avalonswanderer/widevinel3_Android_PoC`

DRM Standardization Privacy Concerns

7

Mmmh mh mh mmmh mh mmmh mh mh mmmh.

Cartographer Cornifer – Hollow Knight

Thanks to HTML5, users can now view videos on Web browsers without installing plug-ins or relying on specific devices. In 2017, World Wide Web Consortium (W3C) published Encrypted Media Extension (EME) as the first official Web standard for Digital Rights Management (DRM), with the overarching goal of allowing seamless integration of DRM systems on browsers. EME has prompted numerous voices of dissent concerning the inadequate protection of users. Of particular interest, privacy concerns were articulated, especially that DRM systems inherently require uniquely identifying information on users devices to control content distribution better.

In this chapter, we fill this gap by investigating privacy leakage caused by EME relying on proprietary and closed-source DRM systems, here we focus on Google Widevine. We conduct empirical experiments to show that browsers diverge when complying EME privacy guidelines, which might undermine users privacy. We find that many browsers gladly give away the identifying Widevine Client ID with no or little explicit consent from users. Moreover, we implement EME Track, a tool that automatically exploits Widevine to break privacy.

This work benefited from the support of the project ANR-22-CE39-0005 DRAMA of the French National Research Agency (ANR).

Takeaway:

This chapter supports the following contributions:

- ➤ We show that browsers misimplementation of Widevine EME can allow to gather unique fingerprints for Android with long-term stability.
- ➤ Found a never-lying User-Agent header for both mobile and desktop as well as a stateful tracking element.
- ➤ We provide arguments against opaque systems within web standards that could be misused by any website to track users online.

The appropriate background to grasp the details of our contribution is provided in:

- ➤ Chapter 2 provides an overview of DRM systems, focusing on their application in content streaming.
- ➤ Chapter 4 brings the necessary information regarding usage and workflow of the EME recommendation needed to understand this contribution.

Contents

7.1. Context and Motivation

The use of the Web for streaming video services has increased tremendously in past years. According to [Ins22], the video streaming market is projected to grow from $473 billion in 2022 to $1,690 billion by 2029. In a business model based on subscription, video services protect their streams via encryption. These video services put the key in a "DRM license". If a user is subscribed, then the DRM license is fetched from the DRM server, and only the related DRM module, called Content Decryption Module (CDM), can get the key and decrypt the video. We call the pair (license server, CDM) a DRM key system. Multiple actors are involved in providing DRM key systems, three of which are distinguished: Microsoft PlayReady [Mic23], Google Widevine [Goo23d] and Apple FairPlay Streaming [App23]. Naturally, different platforms support different DRM key systems. For instance, video streaming is protected by FairPlay on Apple devices (i.e., iOS devices, Apple TV, and Safari on macOS), while Edge on Windows can rely on PlayReady. The lack of cross-platform DRM compatibility was a major reason behind DRM being ultimately abandoned for iTunes Music in 2007.

DRM and the Web are no strangers since users often stream video on browsers. Historically, DRM on the Web was supported in plug-ins for a long time (e.g., Microsoft Silverlight and Adobe Flash). However, the advent of HTML5-based media playback systems encouraged the media industry to make DRM integration more seamless. In 2012, Google and Microsoft partnered with Netflix (a content provider) to propose a "built-in" DRM extension for the Web: the W3C EME [DSWB19], with the overarching goal to define a standard DRM API (Application Programming Interface) that would work across multiple browsers or operating systems on a broad of range of devices. The EME specification does not create yet another DRM key system. Instead, it allows browsers to discover, select and interact with any DRM module automatically. Thus, EME removes the burden of implementing content protection from browsers, whose role is now restricted to only redirecting DRM messages to the right native key system. In 2016, all major browser vendors demonstrated interoperable support of EME, one year before becoming a W3C Recommendation [W3C17].

Standardizing EME received much controversy, as EME stands at the intersection between the desire for freedom of the Internet and the increasing need to protect premium content and services. On the one hand, the EME project received the Emmy Award in 2019 [Tec18]. Both the W3C Director (Tim Berners-Lee) [Ber17] and the W3C CEO (Jeff Jaffe) [Jaf17] strongly defend EME adoption. On the other hand, opponents, such as Free Software Foundation, criticized W3C for standardizing DRM at the behest of a few actors in private industry and qualified the Firefox decision to support it as "shocking" and "unfortunate" [Fou14]. As summarized in [Hég17], privacy received much attention among the voices of dissent.

Problem Statement. Indeed, a DRM key system could cause user tracking because it might manipulate user-specific information that are either disclosed in EME messages or persistently stored on the user device. The starting point of our work is that, as acknowledged by EME, "*privacy cannot be met without the knowledge of the privacy properties of the Key System and its implementations*". Despite the much-raised controversy, little has been done to understand the privacy properties guaranteed by existing key systems, and how they are related to the EME ones. As EME is already widely deployed [Can23], we take a closer look at the privacy leakage of EME implementations in practice. In this chapter, we conduct, to the best of our knowledge, the first privacy evaluation of EME, covering the format of its messages and the impact of browser default configurations on user tracking. In particular, we

explore the tension between the EME privacy guidelines and its actual compliant modules. We only focus on EME as implemented by Widevine for two main reasons. First, Widevine is the most versatile DRM key system, since it is deployed on various operating systems (e.g., Linux, Windows, macOS, and Android) on various devices, ranging from personal computers to smartphones, smart TVs, and embedded devices. The scope of other key systems, such as PlayReady and FairPlay, would have been more limited. Second, each key system introduces numerous proprietary technical details, and we preferred clarity in our presentation and longitudinal analysis over completeness, especially since our conclusion would have remained the same: there is a real risk of privacy leakage caused by EME key systems.

Findings Summary. Our main goal is to gain insights into how the opaque Widevine EME can lead to privacy leaks. In other words, we investigate whether users have any guarantee about their privacy with EME relying on proprietary and closed-source DRM modules like Widevine. We refine our goal into two high-level questions: (1) is there any gap between the EME privacy guidelines and the browser implementations of Widevine-based EME? and (2) what is the privacy impact of this gap? To answer our questions, we first reverse engineer the EME messages as defined by Widevine. Second, we examine the privacy mechanisms put in place by Widevine on Windows, Linux, and Android. We conduct our experiments with a selection of popular browsers as well as privacy-focused ones. Third, supported by our experimental setup and reverse engineering, we find that EME Widevine misses applying some important EME recommendations. In our analysis, we focus on the Widevine distinctive identifier, namely Client ID, and session-related stored data, namely persistent sessions. In that sense, we find that all desktop browsers fail to encrypt the Client ID containing identifying information about the user device. Ironically, we find that some mobile privacy-focused browsers, such as Ghostery, do not encrypt the Client ID, which can be obtained by a few JavaScript calls with no or little explicit users consent, as desired by EME designers [Wat17a]. In addition, we highlight some mobile browsers storing Widevine data as cookies even when first-party cookies are not enabled. Fourth, we continue our study and inspect Widevine device fingerprinting and resulting user tracking. Here, among others, we examine the nature of the Client ID and look deeper into its different fields, uniqueness, and stability. We show that this Client ID allows not only building an augmented User-Agent on which browsers have no control but also a stable and unique fingerprint on Android mobile devices. Overall, our work shows empirical evidence that EME constitutes an effective privacy leakage vector, implying that the W3C claims about "greater privacy" guaranteed by EME are just all red herrings.

Threat Model. By design, DRM systems are bound to some individualization process. Indeed, despite EME recommendations, Widevine establishes a factory provisioning process to embed some unique Device ID on the client side. Widevine defines a privacy mode in which this unique identifier is encrypted before being sent to any remote server. Only Widevine and its partnered service providers, aka Over-the-Top (OTT) platform (e.g., Netflix, Disney+), can decrypt it, implying that they can effortlessly fingerprint all devices. Widevine does not publicly state the required terms for being a partner OTT. This could be concerning, but we consider a wider threat model: an attacker (website) with no Widevine partnership, meaning not having a Widevine-signed server certificate, that can call EME JavaScript API (see subsection 7.5.1 for required permissions) acting as a mirrored proxy between a client and a license server. In this work, we show that this attacker can achieve three goals:

1. Fingerprint users on Android devices if they are on a browser not applying the privacy mode (e.g., firefox).

2. Get identifying, but not unique, attributes if the privacy mode is not always enforced (e.g., all desktop browsers).

3. Achieve surreptitious cross-session tracking on the same origin even when cookies are blocked and site data are wiped. For example, this allows an attacker to link user activity on the same site.

Contributions. We summarize our contributions as follows:

- We reverse engineer the EME messages as defined by Widevine.
- We analyze the browser implementations of Widevine-based EME with respect to the EME privacy guidelines.
- We explore the amount of privacy leakage whenever browsers do not strictly comply with EME guidelines.
- We implement and open-source EME Track; a tool leveraging our findings about EME to demonstrate user tracking.
- Our results were acknowledged by the Mozilla Hall of Fame.

7.2. Background on Web Tracking

This section aims to elucidate the complexities surrounding web tracking, a pervasive issue with significant implications for user privacy. The discussion is organized into two primary categories: stateful and stateless tracking. This background serves as a foundation for understanding the broader implications of web tracking techniques in the context of DRM systems like Widevine.

7.2.0.1. Stateful Tracking

Stateful tracking refers to any mechanism storing data on user devices to identify and track them in the future. The first example of such mechanisms are cookies, which were first introduced to enable a stateful web experience but have been rapidly used to impact privacy by third-party websites [Sch01]. Stored values could then be used across origins to identify users and trace navigation. With this first step, trackers pushed HTTP cookies further using similar storage components like Flash cookies [AWS+11], JavaScript properties, or even HTML5-specific mechanisms [WP12]. By combining these systems, Cookie respawning and Evercookies [Kam10] were introduced to restore previously removed cookies. Using, for instance, Flash cookie, and HTML Storage API, multiple studies have shown its effectiveness in circumventing anti-trackers [SCM+10; AEE+14].

7.2.0.2. Stateless Tracking

In 2009, Mayer [May09] looked into whether the variations resulting from browser attributes could result in the deanonymization of web clients. He checked to see if a distant server could make user identification possible using variations in browsing contexts. He observed that a browser might display values or behaviors linked to the hardware, operating system, and browser setup creating a device fingerprint and enabling user tracking without the need for data to be stored on the user device. Since then, multiple studies have been made on even larger scales to inspect the impact of fingerprinting on web privacy [Eck10; LRB16; GLB18]. These studies have added different attributes to the fingerprint, such as fonts, timezones, JavaScript engine behaviors [MBYS11; MRH+13] or even specific APIs like WebGL or Canvas [MS12], to generate stronger fingerprints both in terms of uniqueness and stability over time.

7.2.0.3. Countermeasures

To circumvent both stateful and stateless tracking, defenses have been put in place to increase user privacy. User tracking through third-party cookies is one of the most used techniques for stateful tracking nowadays. In response, web browsers are now, for most, blocking third parties from using cookies by default. In addition, Google launched in 2022 the Privacy Sandbox [Goo22d] initiative to produce new privacy-oriented web standards to remove third-party cookies and increase privacy boundaries across websites.

With stateless fingerprinting, two opposed trends can be observed; either increasing the fingerprint homogeneity among all users or breaking its stability. The first solution is the one chosen by the Tor browser [Pro22] to blend users into the masses without distinctive attributes. Apple also went this way with the Safari web browser, using it to reduce differences between users [Giz18]. Oppositely, web extensions have been developing to break fingerprinting stability over time, such as Canvas Defender [Mul22], adding noise to the Canvas API calls between browsing sessions.

Another defense chosen by developers is to reduce the browser APIs surface to minimize the information that can be retrieved by origin. Web extensions, such as Canvas Blocker [kka22] or NoScript [Mao22], have been proposed with these goals in mind by reducing APIs for untrusted websites or deactivating JavaScript support to avoid potential fingerprinting. Browsers have also incorporated built-in protection against fingerprinting like Tor or Brave [Bra22b] blocking by default several APIs or JavaScript execution. Firefox has put in place a by default Enhanced Tracking Protection [Moz23] blocking known trackers and fingerprint scripts based on a denylist to avoid negatively impacting users browsing experience.

7.3. EME Widevine

As presented in subsection 2.2.4, Widevine is the most deployed DRM system in the wild and is, therefore, heavily used by web-based content providers through the EME standard interface. In this setting, Widevine specifies the content of the opaque messages in EME for both license acquisition and renewal. Related works dissect the Widevine CDM in Android environments [PSF22b; PSF22a] and desktop ones [Had21; rla22]. In this section, we reverse engineer the EME instantiation by Widevine regarding license acquisition and persistent license. Our goal is not to explore all the underlying cryptographic operations but to study

the content of the opaque messages. In addition, we examine the mechanisms put in place by Widevine to address the various EME privacy concerns.

7.3.1. Reverse Engineering Settings

We conducted a two-stage approach to inspect Widevine-based EME. The first settings are as follows: we leveraged some web extensions that monitor all calls to EME API. Then, we requested media from various content providers, and noticed that EME works similarly for both paid services (e.g., Netflix and Disney+) and experimental ones (e.g., Bitmovin). Finally, we examined all the collected messages to determine which parts are Widevine-defined and which are content provider-defined

As for the second setting, we developed a script calling EME APIs, and hosted it on our web server. Our script implements the workflow in Figure 4.1 of Chapter 4 with Widevine as the underlying DRM system. To this end, we leveraged the integration platform made available by Widevine for external developers to perform license acquisition/renewal [Wid23]. Finally, we accessed our script, while varying operating systems (i.e., Android, Windows, Linux, and macOS) and browsers (e.g., Firefox, Chrome, Edge). This allows us to fine-tune our understanding of different fields that remain unchanged or mutate for a given execution environment.

7.3.2. Opaque Messages in Widevine Workflow

> **Takeaway:** The Widevine protocol and messages bring privacy concerns. Persistent sessions within the protocol could be used to track users if not treated as regular web cookies. The Client ID inside the license request and renewal request could be a distinctive identifier. Therefore, Widevine proposes a Privacy Mode to web browsers and OTT apps to encrypt such data.

As studied in [PSF22a], the Widevine protocol defines four main operations: License/Renewal Requests and License/Renewal Responses. Here, we investigate the correspondence of these operations with the EME protocol. To acquire a media license, the EME user-agent creates a session within the Widevine CDM to ask for a license request through the `generateRequest` EME API. The CDM requires some *Initialization Data* to issue licenses, also stated as *Content Keys*. These data include Content Key IDs associated with some encrypted media. The corresponding request contains a Request ID for associating subsequent responses, one or several Content Key IDs for the desired media, and a Client ID used for license protection (further details are given in subsection 7.3.5). Using the EME JavaScript API, the browser sends this request alongside its signature to a license server, which will respond with the corresponding license response. This response contains the Content Keys for media decryption, alongside key usage control data: Time-To-Live (TTL) and license policies (refer to subsection 7.3.4). Using `update`, the response is then forwarded to the CDM session, which is then ready to decrypt the content.

During their lifetime, licenses can be renewed to extend license TTLs. The CDM can generate a renewal request if the license policy authorizes an extension to the expiration date. In Widevine, this message includes the Request ID of the previous license request and, depending on license policy, the Client ID of the device. The user-agent receives this request as an EME event through the `MediaKeyMessageEvent` to send it to the corresponding server.

Widevine Protocol	*EME API*	*Message Content*
License Request	`generateRequest`	Request ID Content Key ID(s) Client ID
License Response	`update`	Request ID Content Key(s) TTLs License Policy
Renewal Request	`MediaKeyMessageEvent`	Request ID Client ID†
Renewal Response	`update`	Request ID Updated TTLs Updated License Policy

† Optional, presence defined by license policies.

Table 7.1. – Content of Widevine Messages for License Acquisition and Renewal.

The license server responds with an updated TTL, new license policy, and Request ID, all forwarded to the CDM using the `update` call like a usual license response. A summary of the content of opaque Widevine messages can be seen in Table 7.1.

7.3.3. Persistent Session

As explained in section 4.2, EME can define a persistent session type. Once closed, a persistent session can be re-opened to retrieve content keys without the need for a license acquisition phase. In Widevine, the workflow does not differ for a temporary or persistent session; the type is only mentioned when opening the session. However, a persistent session becomes persistent only when a response is processed through the `update` call and the associated license policies permit it. Upon session closing, Widevine entirely relies on the user-agent implementation to store a session blob from the CDM. Here the user-agent is responsible for implementing a per-origin policy as mentioned in EME using, for instance, service storage for licenses inside dedicated databases. Sessions can then be re-opened using their session ID through the `load` function.

7.3.4. Widevine License Policy

License policies are included in several Widevine EME messages. Their goal is to add management rules over content licenses. These rules are defined on the license server side and are enforced by the CDM. A license server can define such policy to control renewal and persistent rights, the timing for renewal requests, license and media playback duration, or rental duration for offline licenses. Among these settings, a license server can ask the CDM to always include the Client ID within license renewal requests. By leveraging the Widevine Integration Platform, allowing a set of policies to be defined by developers for integration testing, we reversed the inline policies sent to the license server to identify various policy settings. The resulting correlation between inline codes and policies can be seen in Table 7.2; note that these codes might not be exhaustive.

Code	*Policy Name*	*Description*
0x08	Can Play	Allow playback.
0x10	Can Persist	Allow the license to be stored for offline usage.
0x18	Can Renew	License can be renewed to extend duration.
0x20	Rental Duration Seconds	Time in seconds for which playback is allowed.
0x28	Playback Duration Seconds	Time in seconds for which playback is allowed once started.
0x30	License Duration Seconds	Time in seconds for which a license is valid.
0x38	Renewal Recovery Duration Seconds	Time in seconds for which playback is allowed until successful license renew.
0x42	Renewal Server URL	URL of the license server for renewal request.
0x48	Renewal Delay Seconds	Time in seconds before trying to renew a license, starting from its usage.
0x50	Renewal Retry Interval Seconds	Time in seconds between renewal requests if renew is unsuccessful.
0x58	Renew with Usage	Try to renew license as soon as it is used.
0x60	Always include Client ID	Force the Client ID presence within renewal request.
0x70	Soft Enforce Playback Duration	Enforce playback duration based on license duration.
0x78	Soft Enforce Rental Duration	Enforce rental duration based on license duration.
0x80	Watermarking Control	Specify a watermarking system.

Table 7.2. – Widevine License Policy Fields

7.3.5. Widevine Client ID

Reminder: Widevine messages incorporate identifiers and key IDs but also a Client ID composed of device-specific information. As discussed in subsection 5.6.1 this could lead to privacy issues discussed in this chapter.

Takeaway: The Widevine Client ID is composed of a certification chain being the Device RSA Key signature chain, and a structure called the Client Info which displays attributes on the device system like version and build info.

In Widevine, the license request includes a Client ID being 40 times the size of the request ID and key ID combined, representing almost the entire size of the request. The Client ID consists of the Client Info and the Device RSA Public Key. These Client Info contain various metadata providing information about device-specific software versions and architectures. Among them, we can find the CPU architecture, OS name, Original Equipment Manufacturer (OEM) Crypto library version number, model name, or the complete device build name. The complete list of Client Info attributes can be seen in Table 7.3. Note that some attributes do not apply to desktop implementations; for instance, desktop Client Info contain the architecture, company, model, platform, and CDM version.

In addition to metadata, the Client ID includes a DRM certificate chain composed of the device DRM public key, called the *Device RSA Public Key*, signed up to Widevine root certificate. On request, the Device RSA Public Key is used by the license server to encrypt licenses based on a Widevine-specific crypto key ladder [PSF22a], while its corresponding private key is used by the CDM to sign license requests. The structure of the Client ID can be seen in Figure 7.1.

7.3.6. Privacy Protection Mechanisms: Privacy Mode & VMP

The inner structure of the Client ID used during the license acquisition of Widevine has brought potential privacy issues correlated to the ones raised by the EME standard. Indeed, the Client ID might constitute a *Distinctive Permanent Identifier*, since it includes architecture, company, build, or even the certificate chain. As an example, Table 7.3 shows the identifying characteristic of a Pixel 7.

To avoid the leakage of such data, Widevine has put in place the *Privacy Mode*. This mode needs to be enabled by the user-agent and forces the CDM to generate a *privacy key* to encrypt the Client ID within requests between the license server and the CDM. A privacy key is generated for each new request and used to encrypt the Client ID using AES in CBC mode with random IV. This key, which is a 128-bit AES key, is then encrypted by a *Server Certificate*, sometimes referred to as *Service Certificate*, and placed within the request. This server certificate is either provided by the default CDM instance configuration or by the license server using the EME API `setServerCertificate`. When provided by a server, its authenticity is verified by a hard-coded public key within the CDM.

Closely related to the Privacy Mode, the Verified Media Path (VMP) is used to verify the integrity of the CDM using cryptographic signatures to enforce a protected video decryption and decoding chain. VMP implicitly mandates the usage of Privacy Mode in desktop implementations [Goo22g]. VMP is supported in macOS and Windows, but not Linux. Android systems rely on hardware protection for secure video decoding. VMP is, therefore, not necessary, leaving the Privacy Mode activation to the user-agents.

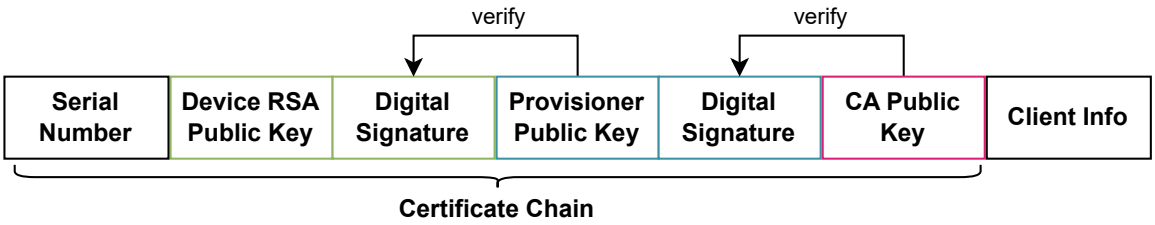

Figure 7.1. – Client ID Fields.

Attribute	*Description*	*Example*
Architecture	CPU architecture	arm64-v8a
Company Name	Device manufacturer	Google
Device Name	Device codename	panther
Product Name	Product codename	panther
Model Name	Device model name	Pixel 7
Platform Name	Desktop OS name	N/A
Application Name	Calling Application	org.mozilla.firefox
Package Cert Hash	Hash of Application Certificate	p4ti...xmwQ=
Build Info	Complete build name	google/panther... 8940162..keys
CDM Version	CDM protocol version number	17.0.0
Security Patch Level (SPL)	OEM Crypto SPL	0
OEM Build Info	OEM Crypto level and build date	OEMCrypto Level3 Code May 20 2022 21:36:542

Table 7.3. – Widevine Client Info.

7.4. Impelementations of Widevine Privacy Mechanisms

7.4.1. Motivations and Research Questions

Recall that the main goal of our study is to gain insights into how the opaque Widevine EME can lead to privacy leaks in practice. To this end, we aim to answer the following high-level questions:

1. Is there any gap between the recommendations in the EME specifications and the browser implementations?
2. What is the privacy impact of this gap?

In this section, we begin by refining the first question above into three research questions (RQs) that guide our search for a better understanding of Widevine EME in the wild. The opaque nature of Widevine makes this task daunting since there is little documentation related to this topic. Thus, we develop a systematic approach to examine how browsers comply with the EME privacy guidelines for Widevine. We detail the second question in section 7.5.

As mandated by EME, the usage of a DRM system shall not reveal any identifying information about the user device. Our reverse engineering of the Widevine EME (refer to section 7.3) highlights the primary sources of privacy leakage in Widevine: (1) the Client ID that contains identifying or unique data per device, and (2) the data stored for persistent sessions, since they might behave as cookies. While Widevine has grown to be a multi-billion dollar industry and is basically available on every platform, the Widevine EME has never been subjected to a comprehensive privacy assessment. Accordingly, our study aims to shed light on the less-understood privacy impacts of Widevine in the context of EME implementations. We present our study that answers the following questions with a methodology involving various empirical evidence and observations. Indeed, we evaluated nine mobile browsers and six of their desktop counterparts, including both mainstream and privacy-focused ones. With our study, we aim to answer the following Research Questions that arise:

- **RQ1:** Do browsers activate the Privacy Mode of Widevine by default for license acquisitions?
- **RQ2:** Does the Privacy Mode protect all occurrences of the Client ID in the Widevine workflow?
- **RQ3:** How does the data related to Persistent Sessions behave like cookies?

7.4.2. Browser Selection

To investigate the privacy mechanisms of Widevine on desktops and Android mobiles, we selected popular web browsers based on their market share, as well as privacy-focused ones according to the user base and download number. The exact version for each selected browser can be seen in Table 7.4.

On desktop, we opt for Chrome, Edge, Firefox, and Opera for mainstream browsers [Cou22a]. For privacy-oriented ones, we got Brave and Tor with respectively 50M and 2.5M monthly users [Tor23; Bra22a].

On Android, we used the Android versions of Chrome, Edge, Firefox, and Opera, to which we added the Samsung Internet Browser to represent the majority of users [Cou22b]. For

Category	*Browser*
Desktop Browsers	Chrome 109.0.5114.120
	Edge 109.0.1518.70
	Opera 94.0.4606.76
	Firefox 108.0.2
	Brave 1.47.186, Chromium 109.0.5114.119
	Tor 12.0
Mobile Browsers	Chrome 109.0.5114.85
	Edge 109.0.1518.70
	Samsung Internet Browser 19.0.6.3
	Opera 73.1.3844.69816
	Firefox 109.1.1
	Firefox Focus 108.1.0
	Brave 1.47.175, Chromium 109.0.5414.87
	Tor 102.2.1
	Ghostery Browser build #2015907851

Table 7.4. – Detailed Web Browser Versions.

privacy-focus ones, claiming enhanced features against user tracking and fingerprinting, we inspected Ghostery and Firefox Focus with 1M+ and 10M+ downloads on the Google Play Store [Goo23b; Goo23a] respectively. In addition, we included the mobile apps of Brave and Tor.

Note that for this study, Safari is excluded since the Widevine DRM is not supported, and replaced when on macOS by Apple own key system FairPlay, being out-of-scope of this study. We do not consider iOS, since Widevine is not supported on it.

7.4.3. Experimental Design

7.4.3.1. Browsers Setup

During our experimentation, we categorized desktop and Android mobile browsers regarding their underlying implementations into two main families: Chromium-based and Firefox-based. This distinction allowed us to establish similarities in the results. In addition, we tested desktop browsers both with VMP-compatible OS (macOS and Windows) and non-VMP, such as Linux systems, for Privacy Mode enforcement testing. Regarding browser configuration, we tested the default settings and privacy-enhancing features when available.

7.4.3.2. RQ1

The Privacy Mode is defined to protect the Client ID. Some might reasonably assume that the Widevine CDM enforces it by default regardless of the underlying browsers or operating systems. Thus, the Client ID is always encrypted, and never appears in clear while being transferred. To challenge this claim, we perform both static and dynamic analyses. First, we rely on our reverse engineering settings (refer to subsection 7.3.1) to ask Widevine to start the license acquisition protocol and intercept the generated license request. Note that we keep the default configurations while making the request. We consider that Privacy Mode is not

enabled by default if the obtained Client ID is not encrypted. Second, we statically inspect browsers binaries and source code when available to search for any Widevine Privacy Mode related calls. In particular, Android allows explicit activation of the Privacy Mode through its DRM API. We look for these corresponding calls. To the best of our knowledge, there is no equivalent call on desktops.

7.4.3.3. RQ2

The Client ID is present in the license request, which is encrypted when the Privacy Mode is enabled. Based on Table 7.1, the Client ID can also be present in the renewal request. Therefore, we answer our question by making Widevine generate license renewals to look into their content for the Client ID. Unlike license requests, EME does not provide a public API to create renewal requests. Indeed, anticipating the expiration of a key, the Widevine CDM triggers an event including the renewal request to be forwarded to the license server. The problem is that the Widevine test license server in our experimentation settings provides licenses that do not issue any renewal event. In order to overcome this limitation, we leveraged once again the Widevine Integration Platform. Recall that for testing purposes, this platform allows us to define license policies through an argument within the license server URL. Thus, we specify the policies so that a renewal request is generated a few seconds after loading the related license. We also mandate the optional Client ID to be included in this request. Finally, we automate our script to recover the license and renewal requests from the EME workflow that is supported by numerous browsers in various environments. This question evaluates the adequacy of Privacy Mode implementations in protecting the Client ID.

7.4.3.4. RQ3

To address this question, we need to open persistent sessions by the Widevine CDM through custom license policies allowing offline license storage. Here, we verified three properties related to cookies. First, we check whether persistent sessions can be opened even when first-party cookies are blocked. Second, we attempt to re-open the closed persistent sessions from different origins and browsing profiles. Third, we study the impact of browsers deleting cookies and site data on the stored data. On mobiles, we went further and deleted all the data via the Android app settings. Ideally, to follow EME recommendations, a persistent session is bound to an origin, only supported when first-party cookies are allowed, and cannot be re-opened when cookies are cleared.

7.4.4. Results

> 🔑 **Takeaway:** On desktop, web browsers only rely on VMP to activate the Privacy Mode leaving the Client ID in clear. When available, persistent session data are treated as regular web cookies. On Android, Firefox-based browsers do not implement the Privacy Mode of Widevine, leaking the Client ID as well. For Chromium-based browsers, two of them do not treat persistent sessions as cookies.

Here, we cover the findings obtained while following our methodology to answer our three research questions. A summary can be found in Table 7.5. Tor and Brave mobile do not support EME, and therefore we were not able to perform any relevant experimentation on them. Thus, they are noted as N/A in our table.

Browsers	RQ1		RQ2		RQ3
Desktop	**VMP**	**non-VMP**	**VMP**	**non-VMP**	
Chromium Family					
Chrome	●	○	○	○	●
Edge	●	○	○	○	●
Opera	●	○	○	○	●
Brave	●	○	○	○	N/A
Firefox Family					
Firefox	●	○	○	○	N/A
Tor	N/A	N/A	N/A	N/A	N/A
Android Mobile					
Chromium Family					
Chrome	●		●		●
Samsung	●		●		○
Edge	●		●		●
Opera	●		●		○
Brave	N/A		N/A		N/A
Firefox Family					
Firefox	○		○		N/A
Firefox Focus	○		○		N/A
Ghostery	○		○		N/A
Tor	N/A		N/A		N/A

● EME compliant.
○ Do not respect EME privacy recommendations.

Table 7.5. – Results for Implementation Questions per Browsers.

7.4.4.1. RQ1

As presented in Table 7.5, we can see that the Privacy Mode and VMP are strongly bound to each other on desktop. Indeed, the Privacy Mode is enabled whenever the VMP is available on the underlying operating system. Thus, Privacy Mode is activated on Windows and macOS regardless of the browser used. In contrast, no browser protects the Client ID in license requests on Linux where VMP is not yet supported, as indicated by Widevine [Goo22g]. We conclude that the Privacy Mode only depends on the Widevine CDM that checks VMP before enforcement. No further action is required by browsers, explaining their identical behaviors.

As for mobiles, we find that browsers behave differently, and can be categorized into two families: Chromium and Firefox. The Chromium family includes Chrome (of course), Samsung Internet Browser, Edge, and Opera. As for the Firefox family, it includes Firefox, Firefox Focus, and Ghostery. Our experiments show the Client ID in clear for the Firefox family, implying no Privacy Mode. Only encrypted license requests were observed within the Chromium family. We push our analysis further to understand the rationale behind this divergence. Indeed, we reverse engineer browser apks with Jadx [Sky19] to identify any calls to the `MediaDRM` class, which is the Android component communicating with the Widevine CDM. In particular, we spot the use of the `setPropertyString` method with the parameters `('PrivacyMode', 'enable')` for the Privacy Mode. Enforcing our claims, we encounter the call to this `setPropertyString` method only in the Chromium family. It is almost ironic that privacy-centered browsers, such as Firefox Focus and Ghostery, do not enable the Privacy Mode, thereby leaking sensitive and identifying data through the EME API.

7.4.4.2. RQ2

In this question, our methodology is to continue the EME workflow started previously as we force the event renewal following the license request generation in RQ1. Then, we investigate whether the Client ID is encrypted in the renewal request. Our evaluations complete our observations in RQ1. Indeed, we first observe that whenever the Client ID is not protected in the license request, it always appears in clear within renewal requests regardless of the execution environments. This concerns Linux systems on desktop and the Firefox family on mobile, and comes in accordance with the RQ1 findings. Second, we explore browsers on VMP-compatible systems. Once again, we notice that all browsers behave in the same way, strongly implying that the observed behavior is implemented by the CDM. Here, we witness the absence of encryption regarding the Client ID within the renewal request. Third, we analyze the mobile Chromium family. Our tests conclude that the Privacy Mode is also applied to the renewal event; the property set using the `MediaDRM` class enforces the Privacy Mode on all Client ID occurrences.

7.4.4.3. RQ3

Unlike the Privacy Mode, browsers differ greatly regarding persistent sessions. Recall that we study three properties: (1) preventing cross-origin sessions, (2) activation when blocking first-party cookies, and (3) availability after wiping cookies and site data. Here, we needed to exclude some browsers, since they do not provide any support for persistent sessions: the Firefox family on mobile as well as Firefox and Brave on desktop.

As for the examined properties, all browsers satisfy the first property, namely that sessions are only accessible by the origin, creating them from the same browsing profile. This implies that no malicious website can leverage sessions from other valid OTTs. Regarding the second property, we find that almost all browsers prevent persistent sessions when first-party cookies are blocked. The only exceptions are the mobile Opera and the Samsung Internet Browser. The same trend carries on for the third property. Indeed, only mobile Opera and the Samsung Internet Browser do not remove the persistent sessions after deleting cookies and site data, here note that erasing cookies cannot be done without erasing also site data from the browser settings. In fact, following our methodology, sessions can still be re-opened after wiping data for these two browsers. However, persistent sessions are indeed removed when clearing application data from the Android app settings.

7.4.5. Implications and Insights

Nowadays, the EME standard is supported by almost all browsers [Can23], implying that any privacy leakage might impact a large user base, represented by our browser selection explained in subsection 7.5.2. The W3C has sought to reassure the web community by minimizing such a risk. Tim Berners-Lee stated that "*the EME system can sandbox the DRM code to limit the damage it can do to the users privacy*". In that sense, the EME specification includes various guidelines to enforce user privacy. Our experiments show that these guidelines are not technically enforced in practice. Indeed, the opaque nature of EME makes it hard to assess any privacy properties related to the use of a given proprietary module. More importantly, EME states that "*user-agents must take responsibility for providing users with adequate control over their own privacy*", while allowing DRM providers to define their own format key contents that might involve distinguishing identifiers.

The EME specification (refer to section 11.4.2 of [DSWB19]) presents their recommendations for a privacy-friendly EME implementation. Of particular interest, we mention two recommendations: (1) encrypt distinctive identifiers, and (2) treat key systems stored data like cookies or web storage. However, we argue that these considerations lack binding requirements and are insufficiently concrete. In that regard, users do not have any guarantee that their privacy is safeguarded when using EME with proprietary DRM modules.

In this chapter, we empirically challenge these privacy claims by scrutinizing one of the most popular real-world underlying CDM of EME, namely Widevine. We provide, to the best of our knowledge, the first conformance test verifying EME privacy recommendations for Widevine EME. Supported by our experimental setup and reverse engineering, our findings show a gap between these recommendations and deployed implementations of Widevine EME, implying a rather concerning scenario.

First of all, we find that all desktop browsers fail to encrypt the Client ID, which represents distinctive characteristics in the Widevine protocol. Indeed, the Widevine CDM does not enable the Privacy Mode whenever VMP is not supported (e.g., Linux). Surprisingly, the CDM does not apply the Privacy Mode on renewal requests even when VMP is supported. On mobiles, privacy browsers do not encrypt the Client ID, although the Android platform provides all the necessary API for this purpose. Only the mobile Chromium family never reveals the Client ID in clear. As for the persistent sessions, the desktop browsers perform better, since the related data cannot be stored when the cookies are not allowed, and are always deleted when cookies are cleared. On mobile, the Chromium family acts differently. Chrome and Edge operate likewise their desktop counterparts. However, Opera and Samsung do not treat the stored data like cookies or web storage. For instance, there is no way to delete persistent session related data other than wiping all browser data from the Android app configurations. We notice that persistent sessions are not supported on all EME browsers: Firefox and Brave desktop do not offer such support. Overall, when supporting persistent sessions, browsers generally tend to comply with the EME specifications by managing related stored data as cookies. Otherwise, such data can be leveraged to track users having no easy mechanism to prevent their storage or even delete them.

7.5. Privacy Implications

Our findings in section 7.4 point to a gap between EME and browsers in terms of privacy. In our analysis, we focus on the Client ID, aka distinctive identifier, and persistent sessions, namely session-related stored data. We continue our study to answer our second question: what is the privacy impact of this gap? Here, we refine our question to inspect device fingerprinting and user tracking. First, we examine whether the Client ID certificate chain constitutes a device fingerprint uniquely identifying the device starting EME. Second, we consider the Client Info attributes (refer to Table 7.3) and compare it with the User-Agent HTTP Header (UA) provided by browsers. Third, We discuss how persistent sessions can be leveraged for user tracking even when cookies are blocked or cleared. Below, we provide answers about each investigation. However, we start by reviewing the permissions required to perform the described actions; namely, we probe the permissions a website needs to call the EME API. We also present our experiment setup in a second time.

7.5.1. Permissions

Our methodology is simple: we view some DRM-protected content on different browsers fresh installation and note whether users are asked to explicitly validate any related permission. Again, the Firefox and Chromium families behave differently. In the Chromium family, content protection service is allowed by default for all origins, leaving the user unaware of EME access by the web page. Regarding the Firefox family, permission is explicitly required for DRM access, but differs from desktop and mobile versions. On Android, permission is asked once per origin trying to access the EME API, without key systems distinction, while for desktop browsers, this permission is granted once for all origins and key systems. This implies that a desktop user granting DRM access to a legitimate OTT platform is also allowing illegitimate web pages to access EME. Firefox asks for permission by displaying the following message: Allow <URL> to play DRM-controlled content? With two buttons, "Don't allow" and "Allow" with "Allow" pre-selected.

7.5.2. Experimental Setup

To explore our question, we deployed our tests on multiple devices both for desktop and Android. Note that for both experiment setups, no real users were involved avoiding user privacy risks.

On Desktop. For our population, we used 159,923 devices with Windows 10/11, macOS Big Sur and Linux. The majority comes from Windows and Mac VMs of BrowserStack[1], added to our own device pool of Windows and Linux devices for comparison purposes between real and emulated devices. To inspect Widevine Client IDs and persistent sessions, we automated navigation to our custom site in which we open persistent sessions when available, and perform license renewal to obtain clear Client IDs using custom license policies as described in subsection 7.4.3.

On Android. We used 317 physical devices from BrowserStack and our own device pool, as well as 18,101 emulated ones, to collect Client IDs. We developed an Android app that communicates with the Widevine CDM, as a browser would do, to generate clear license requests and gather Client IDs in a fully automated way. For emulated Android devices, we used Google Pixel 1 to 6 from Android Studio [And23] with factory settings using Android version 8.1 up to 13.

Emulated devices. As mentioned in [CLB+22], using emulated devices for web tracking measurement could have an impact on results relevance. Obviously, our created population cannot be used to represent a real-world community; for instance, specific phone models and Android versions might be over-represented. However, we took great care to avoid that our results would be biased regarding the uniqueness of the collected identifiers. Indeed, the Client ID contains the RSA public key of the key pairs sent by the provisioning server. As pointed out in [PSF22b], the provisioning server generates a distinct Client ID for each Device ID. Provisioning with the same Device ID results in the same Client ID for the same app. Therefore, for license individualization on physical devices, Widevine defines a factory provisioning process to enforce the uniqueness of Device ID on Android devices. For our

[1] available on the platform `https://www.browserstack.com`

experimental settings, we built our population of Android-emulated devices, so that each emulator has a unique Device ID, which is the default settings of the Android SDK tools. As for Desktop, the Client ID solely depends on the CDM version, regardless of the execution environment.

7.5.3. Client ID Certificate Fingerprint

> **Takeaway:** The Client ID certification chain used by Widevine can be used to track users online on Android devices. While non-unique and non-stable on desktop, the mobile provisioning of the Client ID is unique and heavily stable over time.

In subsection 7.3.5, findings from our reverse engineering strongly indicate that the Client ID contains multiple distinctive permanent identifiers. Indeed, its inner Client Info contains device-specific attributes related to hardware instances and software versions for desktop and Android. In addition, it contains an RSA certificate chain used to protect the transmitted licenses. On the one hand, attributes can help classify devices into subgroups. On the other hand, the Widevine certificate chain might be enough to point out an exact device. The nature of this chain depends on the platform as explained above. On desktop (i.e., Linux, macOS, and Windows), the certificate chain is hard-coded within the CDM. Due to this design, all desktop Widevine CDMs share the same DRM certificate chain for a given CDM version. Thus, it does not enable fingerprinting on desktop. On Android devices, a Widevine-specific factory Root-of-Trust (RoT), called the *Keybox* [Wid; PSF22a], is used to perform a *provisioning phase* requesting a certificate. This phase occurs once per app. The received certificate chain seems specific per device (or more precisely, Device ID).

Below, we conduct more experiments to explore two important features: the uniqueness and stability of the Client ID as a potential fingerprint. We only consider the certificate chain part of the Client ID.

7.5.3.1. Uniqueness

The purpose of a fingerprint is to completely characterize a specific device. Therefore, an important related feature is uniqueness: to what extent can we affirm that two license requests come from the same device if they contain the same chain? Here, we performed a clean installation and factory reset of devices for each test to evaluate certificate uniqueness in a *multiple devices, single provisioning* approach using our EME website and custom Android app as explained in subsection 7.5.2.

Our results are twofold. First, as expected, the Client ID is not unique on desktop environments; we only observed two different certificate serial numbers during our tests due to the different CDM versions available between Windows and macOS/Linux. Second, the certificate chain is unique for Android devices. Indeed, We can precisely distinguish all mobile devices in the test set by extracting the certificate serial numbers of the gathered license requests. This implies that an Android device can be tracked by any web origin using the EME API when the Client ID is in clear, which is the case for all browsers among the Firefox family as shown in subsection 7.4.4. We agree that our experiments do not constitute a definite proof of perfect uniqueness for an arbitrary device population, since this would have required a much larger test pool. Nevertheless, we argue that the technical details presented in [PSF22a] regarding provisioning make us believe that this fingerprint is unique by design. In fact, the

Client ID contains the RSA public key of the key pairs sent by the provisioning server. To enforce individualization, the key pairs are not shared among other devices, confirming our findings.

7.5.3.2. Stability

Recall that a unique fingerprint is not enough to track users. The leveraged fingerprint should resist changing with time in order to allow long-term tracking. For instance, a software-based fingerprint changes its value when the underlying software is updated, in contrast with a hardware-based one that is harder to collect. Here, we show that our fingerprint gets the best of two worlds: it is easy to collect (a few EME calls without explicit permissions) and stable over time. Here, we tested the stability of Client ID certificate with a *single device, multiple provisioning* approach by removing the certificate instance to force new provisioning. On fresh settings, the CDM asks for certificate provisioning to create a license request. For desktop, forcing this behavior translates into removing the CDM library from the browser repository, while on Android, removing the stored certificate from the OS file system using privileged user rights, namely the `media` user level. We installed any CDM update when available, but we did not systematically attempt to install new applications for each provisioning.

Similar to the uniqueness results, in desktop implementations, the certificate of the Client ID depends on the version of the CDM. During Widevine updates, the certificate serial number changes for all implementations, meaning stability on a version lifetime, which is 6 months in average. On Android, we adopted two longitudinal approaches: one device over two years, and multiple devices over one month. In both approaches, we removed the certificate to force triggering a new provisioning phase. In the former approach, we recovered the Client ID and removed the certificate at different intervals: one year, 6 months and 3 months. As for the latter, the deleting intervals are shorter: one week, one day, and one hour. After removal, we always get the same certificate, which strongly implies the stability of the Client ID.

7.5.3.3. Comparative Analyses

In order to put our results in perspective, we compare ourselves to three of the most large-scale studies of device fingerprinting on the web: Panopticlick [Eck10], AmIUnique [LRB16], and Hiding in the Crowd [GLB18]. These works inspect the uniqueness of fingerprints on mobile and desktop devices by gathering various attributes available through HTTP headers, plugins, and web APIs. Panopticlick, with 470,161 desktops, and AmIUnique, with 105,829 and 13,105 desktop and mobile devices, have shown a high proportion of uniqueness among gathered information with at least 81% up to 94,2% of unique fingerprints.

Later, Gómez-Boix et al. pointed out, with Hiding in the Crowd, that these results might be biased due to the tested population being privacy-aware users conscious of the collected values by the experiments. By fingerprinting 1,816,776 desktops and 251,166 mobiles, they found uniqueness of up to 35,7% at best.

For our tests, we took the same order of magnitude for the number of devices as AmIUnique. Unlike AmIUnique, our experimental settings consist mainly of emulated devices. As explained previously, we took great attention to build our population, so that the Widevine provisioning ecosystem would process our requests similarly as it does to physical devices. While never unique for desktops, Client IDs of mobile devices offer a distinctive identifier with maximum uniqueness and stability through a JavaScript API available by any origin.

7.5.4. Client Info as User-Agent Header

> **Takeaway:** Within the Client ID, the Client Info structure can be used to gather additional tamper-proof info on the OS and device versions. While non-distinctive, this structure can provide information to an attacker to pinpoint device-specific exploits to use.

As shown subsection 7.3.5 and Figure 7.1, the Client ID includes not only the fingerprint certificate chain, but also multiple other attributes within the Client Info structure. Here, we investigate the privacy leakage of the latter. The Client Info attributes include the CPU architecture, OS name, device name, and the complete device build name. Similar to the User-Agent HTTP Header (UA), these attributes can build a characteristic string for servers to identify the operating system, vendor, and version of the requesting browser or CDM. This is particularly interesting to get an augmented UA in which fields cannot be modified by some user-controlled browser configurations. Recall that the term *user-agent* in EME refers to the software implementing the EME API (i.e., a browser). To avoid confusion in this subsection, we will use UA to refer to the browser User-Agent HTTP Header.

Interestingly, the UA and the Client Info contain redundant information, such as the platform and architecture for both desktop and Android, or even the browser package certificate hash for mobile devices. Based on the UA structure [Mdn23], this redundant data covers most of the UA:

```
1  UA:   Mozilla/5.0 (<system-information>)
2        <platform> (<platform-details>)
3        <extensions>
4  e.g.: Mozilla/5.0 (Linux; Android 13; Pixel 4a)
5        AppleWebKit/537.36 (KHTML, like Gecko)
6        Chrome/113.0.0.0 Mobile Safari/537.36
```

The resulting redundancy can be leveraged to detect browser instances attempting to hide or tamper with the UA in order to access specific resources (e.g., the mobile version of a website). This is possible since any UA modification, by spoofing for instance, has no impact on the Client Info attributes returned by Widevine. Therefore, Client Info can play the role of a never-lying UA. The reliability of Client Info is appealing for services relying on UA as a second-factor authentication [LISP22]: lying users can be detected whenever the purported UA conflicts with the device class indicated by Client Info. Finally, minimizing the UA sent to origins is set as a goal for various privacy efforts, such as the Google Privacy Sandbox User-Agent Reduction proposal [Goo22e]. The fact that some rich UA can be built from Widevine might limit the benefit of these current efforts.

7.5.4.1. Uniqueness

As explained in subsection 7.5.2, our dataset of mostly emulated devices cannot be relied upon to calculate the entropy of Client Info for a real-world population. Nevertheless, we can still provide some insights from our collected data. On Android, the data from Client Info can point to an exact device model and build version using the Build Info string (refer to the example of Pixel 7 in Table 7.3). On desktop, although with fewer fields than on Android, Client Info can still narrow down devices to architecture and OS subgroups, making entropy close to a standard browser UA regarding system information, which was studied in [GLB18].

In the Chromium project, the related Widevine implementation indicates eight possible values[2]: Windows (x86, x64, and arm64), Linux (x64, arm, and arm64) and MacOS (x64, and arm64). It is important to note that Client Info is always obtained with the certificate chain of the Client ID, making its uniqueness less significant, especially on Android.

7.5.4.2. Stability

The Client Info stability depends mainly on the version of the Widevine CDM. On desktop, the available attributes are the OS and architecture being fixed. This leads to strong stability over time for desktop environment. On Android, the Client Info is influenced by the device build version that may change with the Widevine version, patch level, and build info, which makes the Client Info stability dependent on both Android and Widevine updates. The frequency of these updates depends heavily on mobile manufacturers; varying from monthly for Google Pixel phones and never for outdated devices no longer receiving any update. It is worth noting that stability is less relevant for Client Info when it is used to collect identifying data about devices, rather than fingerprinting.

7.5.5. Persistent Session Tracking

> **Takeaway:** Persistent sessions in Widevine enable covert tracking across visits, bypassing cookie restrictions if not implemented as such.

Recall that all browsers from the Chromium family on Android and desktops support persistent sessions in Widevine, except for Brave. This support allows any web origin with EME API access to create a persistent session and collects its session ID for future usage. Persistent session IDs are random 16-byte identifiers represented as a hex string. When a client visits a website, it can be asked to open a previous session with persistent type using its ID; if the operation succeeds, this means that the client has already visited the website before. Thus, any website can check for the presence of a given session ID, revealing previous visitors. The approach is straightforward: attempt to open all the noted session IDs until a success occurs. This is possible because the number of failures is not limited. The scalability of this approach can be improved by breaking down the number of tested session IDs by device categorization. The privacy impact of re-invoking prior sessions is to achieve surreptitious cross-session tracking on the same origin even when cookies are blocked and site data are wiped (Android Opera and Samsung). For example, this allows an attacker to link user activity on the same site.

Based on our findings, persistent sessions are treated the same way as cookie data within browsers as mandated by EME, except for the mobile browsers Opera and Samsung Internet Browser. Oddly enough, they are not displayed in any Storage User Interface within navigator settings. We argue that this might mislead some users unaware of this mechanism storing cookie-like data. As a limitation compared to Client ID fingerprinting, this tracking mechanism is stateful. User tracking through persistent sessions requires the origin to keep track of session IDs due to the lack of API that lists stored sessions within the browser file system.

[2] `https://chromium.googlesource.com/chromium/src/+/HEAD/third_party/widevine/cdm/widevine.gni#23`

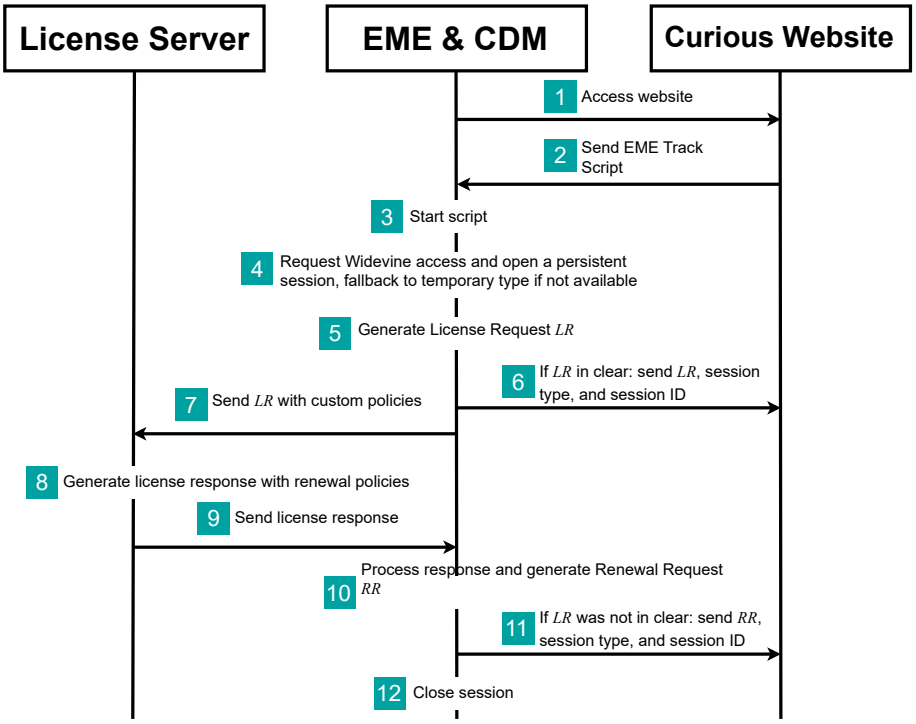

Figure 7.2. – EME Widevine Fingerprint Flow.

7.6. EME User Tracking Scripts

In sections 7.4 and 7.5, we discussed how these EME misimplementations could be exploited to track users either by a stateful cookie-like token or by a unique fingerprint with strong stability. In this section, we detail how a curious website could use these mechanisms to track users over the Internet based on our threat model described in section 7.1. In particular, they set renewal request intervals, force Client ID within EME renewal messages, or even open persistent sessions. To do so, we developed a tool in JavaScript code that sends the Client ID of a user-agent to a malicious server and opens persistent sessions when supported. The script is available on our GitHub[3] for reproducibility purposes. More importantly, our tool can test the conformance of implementations with EME privacy concerns. In the following, we present our workflow depicted in Figure 7.2.

7.6.1. EME Track

Like our experiment settings, we leverage the Widevine staging license server with custom Widevine policies support to operate.

From Figure 7.2, in steps 1 and 2, a user visits a curious origin and receives the EME Track script. The user-agent starts executing it 3, and is asked to instantiate its underlying Widevine CDM with a default configuration with persistent license type support. If not supported, temporary type is used for sessions 4. Here, we assume that users enable EME-related permissions (according to subsection 7.5.1).

[3] https://github.com/Avalonswanderer/widevine_eme_fingerprinting

In step 5, initialization data are provided to Widevine through the `generateRequest` API. These data are related to Widevine pre-encrypted media, available with the staging license server. When the user-agent is in Privacy Mode by default, a service certificate is required and provided by the script to the CDM. Here, this certificate corresponds to the `license.widevine.com` domain. Once processed by the CDM, the initialization data are used to create a license request, noted *LR*. If *LR* is in clear, we forward it in 6, with its session ID and type, to the curious website. The workflow continues from here to complete a standard license provisioning by sending *LR* to the Widevine license server. While doing so, custom Widevine policies are used (see subsection 7.3.4), requiring persistent rights, renewable license, and renewal attempts within a few seconds for the demanded license.

Steps 8 and 9 correspond to the license server response with appropriate content key(s) and rights. The user-agent provides the response to the CDM in step 10, allowing the session to be saved on the file system if persistent. Within a few seconds, license policies force the Widevine CDM to generate a renewal request *RR* including the Client ID of the device. If the previous *LR* Client ID was not in clear text, the user-agent sends *RR*, session ID, and session type to the curious site in step 11 before closing the session in 12.

With either *LR* or *RR* in clear text, a malicious website can gather unique fingerprints with device-specific attributes for mobile devices and a never-lying UA for desktop implementations. With the support of persistent licenses, the received session ID can be used to track users if requests are not distinctive, as shown below.

7.6.2. Persistent Session Re-invocation

With our EME Track script, an origin can gather Client IDs in clear when available from any user-agent and create persistent sessions when supported within the browser file system. Here we describe the workflow of a curious website tracking users based on their stored Widevine persistent sessions.

At first, a user-agent visits a curious website and receives a tracking script starting to be executed. A Widevine access is demanded using `requestMediaKeySystemAccess` with persistent license support configuration. At this point, either by hard-coded values within the script or by receiving it through the origin, previously gathered session IDs are provided by the curious website to the CDM in an attempt to `load` an already existing session. On return, the script provides the result of this operation to the curious website. In the event of user-agent revisiting the website, a persistent session can be opened, allowing the origin to identify the user.

7.6.3. Privacy Leakage in Practice

The adoption of EME by W3C has been a controversial decision [Hal17]. Much of the raised debate was related to the opaque design of EME in which no technical guarantees can be given about the security and privacy properties of an EME-compliant system. It is true that, technically, like any software, EME has the potential to be privacy-invasive and to have possible security issues. However, we have shown that the controversy is well-founded due to the privacy concerns inherent in uniquely identifying keys in Widevine.

The answers to our two high-level questions are: (1) yes, there is a gap between the promised guarantees by EME and the actual Widevine implementations, (2) and this leaks private information, allowing long-term user tracking. Indeed, the Client ID is not appropriately

protected and can be recovered in most browsers. This Client ID allows not only building an augmented UA on which browsers have no control, but also a stable and unique fingerprint on mobile environments. Moreover, EME might store some persistent offline data, that can be invoked subsequently to track visitors. All browsers mislead users by not displaying these EME data in any Storage User Interface. Some browsers do not even include them when wiping site data. More importantly, we show how easy it is to leverage EME to leak data, especially since browsers mostly adopt calling EME "*without annoying and confusing consent prompts*" [Wat17a].

7.6.4. OTTs as EME Actors

> **Takeaway:** Netflix and Disney+ introduce OTT-specific fields in EME messages, complicating EME compliance and adding privacy risks. Netflix ESN, while encrypted, could serve as a unique identifier, exposing users to potential privacy loss.

Section 7.3 and Table 7.1 show the content of EME messages when used with Widevine. During our tests, we observed EME workflow from various legitimate OTTs: Amazon Prime Video, Netflix, and Disney+. We found out that in addition to these arguments, Netflix and Disney+ were using an OTT-specific field within license response and renewal request/response. OTTs can leverage their own license servers to use this field and store information within the client CDM as they please to, for instance, manage communication between client and server statelessly. However, this particular usage changes the EME compliance chain by introducing another actor: OTTs themselves. With such a field, another opaque layer is added to EME messages increasing privacy risks in the event of identifying data. The complexity of involving OTTs chosen data within EME-specific messages is illustrated with Netflix.

When considering the EME messages generated by Netflix, we observe the usage by Netflix of an OTT-specific field in the license response, renewal request, and renewal response. Our experiments show that this field includes the required Movie ID and the device Equipment Serial Number (ESN) defined by Netflix. According to [Blo22], the Netflix ESN (NESN) is used in production to send updates to a specific pool of devices to beta-test new fixes and features. This NESN is generated by appending several data [zac22]: device category (e.g., Chrome browser, aka ChromeCDM for all desktops, Smartphones, Tablets, Android TVs, Smart Displays, Google TVs), the version of the OEM Crypto Library, the manufacturer and the model of the device followed by a random string that is constant for a given device. Therefore, similar to the Client ID, the NESN should be encrypted to protect users privacy in case a proxy is used while streaming.

Thus, we extend our research questions and investigate NESN leakage: (1) is it encrypted? and (2) how does the NESN constitute a fingerprint? While focusing on Netflix, we consider the same experiment settings as described in subsection 7.3.1. It is worth noting that, in license response, Netflix transmits licenses that require renewal a few seconds later. Therefore, we were able to easily observe the whole EME workflow without particular policies from the user-agents context. For both desktop and mobile, from the point of view of EME, the NESN is never protected on license and renewal responses. Nevertheless, Netflix EME messages containing this NESN are always encrypted through the Message Security Layer (MSL) protocol, leveraging Widevine as a cryptographic library, therefore not relying solely on HTTPS. As a matter of fact, the Web Cryptography API [Wat17b] on browsers has been specifically standardized by Netflix within the W3C for this purpose [Net20]. We

conclude that Netflix attempts to keep the NESN private by masking EME messages over the network. On mobile, Netflix only runs with its own application, with no browser support; however, the presence of this NESN within EME-specific messages in desktop implementations involves the OTT in the protocol consideration for privacy. The MSL being out of the EME recommendation, this encryption does not comply with privacy concerns by adding the NESN as a distinctive identifier within opaque messages.

Concerning the second question, we examine the privacy loss from a possible NESN leakage. As often in our study, the fallout depends on whether the leakage occurs on mobile or desktop. For mobile, the NESN ends with a 64-byte string. This string is unique per device for a given Widevine security level (i.e., L1 and L3). In addition, we note that the related fingerprint is stable since it is preserved even after wiping all app data using the Android settings. For desktop, the ending string is composed of 30 bytes. Unlike the Client ID, the NESN is unique per device, thereby might constituting a fingerprint if leaked through EME before MSL usage. Fortunately, the desktop NESN value is linked to cookies and updated after cookies and site data deletion, falling back to standard cookie concerns. We performed our tests on Windows, macOS, and Linux.

Our study of Netflix shows that the privacy guarantees of EME are quite brittle. Indeed, it depends on the implementations of three parties that are proprietary and reluctant to communicate: web browsers, DRM systems, and streaming services. The Netflix case is an example of how complex the streaming ecosystem is. Netflix defines an identifying ESN and argues that it is necessary to beta test its new features, thereby compliant with General Data Protection Regulation (GDPR). However, at the same time, this same ESN brings a distinctive identifier, within the EME protocol, under the responsibility of the OTT.

7.6.5. License Server as Big Brother

> 🔑 **Takeaway:** Even with Privacy Mode enabled, Widevine design allows license servers to access the Client ID, undermining the privacy protections it purports to offer when considering curious license servers.

The EME standard specifies that CDM "*implementations should avoid use of distinctive identifiers*", otherwise, "*the CDM vendor may be able to track the activity of the user*". Despite this recommendation, Widevine defines and shares the Client ID, which encompasses some fingerprinting properties. As explained in subsection 7.3.6, Widevine attempts to limit the damage by introducing the Privacy Mode in which, if enabled, the Client ID gets encrypted with a public key provided by the license server. Recall that OTTs can provide their own Widevine-signed server certificate using `setServerCertificate`. Consequently, these OTTs can get the Client ID of any device, with considerable privacy implications on Android mobiles due to a unique and stable fingerprint.

Some might argue that sharing such a distinctive identifier is vital for DRM functioning to bound licenses to some individualization process. However, this violates users privacy, given the fact that all of this is done without users consent in an opaque way. We are concerned that Widevine and OTTs, via their license servers, are able to obtain identifying information on any device. Indeed, EME states that "user-agents must take responsibility for providing users with adequate control over their own privacy". Our work shows that such a statement about EME respecting user privacy is misleading and is contradicted by the real-world operating of Widevine. Browsers have no choice but to share the Client ID

with license servers whenever EME is enabled, breaking privacy in numerous contexts. We appeal that it becomes compelling for proprietary DRM systems to take responsibility and transparently enforce all EME design suggestions, especially for privacy and security.

7.6.6. Responsible Disclosure

Our findings have been timely communicated to all concerned parties following responsible disclosure processes. Mozilla Firefox was quite responsive, and we got rewarded via their bug bounty program. The Mozilla EME team investigated our findings and released a patch to address the identified privacy issues and acknowledged us in the Mozilla Hall of Fame. Regarding Client ID being in clear in renewal requests, we first contacted the EME Chrome team that reviewed our disclosure report and showed concerns about its privacy consequence, namely the EME user-agent. They confirmed our intuition that the problem is caused by the Widevine CDM. Therefore, we filed a Widevine bug report about missing Privacy Mode on VMP systems, and are still in communication with them.

Formal Verification of Widevine EME

8

We did this not because it is easy, but because we thought it would be easy.

A Wise Mushroom

In this chapter, we take a first step into the formal verification of the Widevine Digital Rights Management (DRM) system. Despite its widespread adoption, facilitated by its integration with the W3C Encrypted Media Extension (EME) API, the formal security aspects of Widevine remain largely unexplored. To address this gap, we employ the Tamarin prover, a formal verification tool, to dissect the Widevine protocol within the EME context. We define explicit security goals that DRM systems should aim to achieve and use Tamarin to prove these goals within the Widevine-EME framework. This chapter serves as an initial attempt to present the formal security guarantees of DRM systems like Widevine.

Takeaway:

This chapter provides the following contributions:

- A full overview of the Widevine protocol under the EME API.
- The definition of high-level security goals for DRM.
- Presentation of ongoing Tamarin representation of the EME Widevine protocol.

The appropriate background to grasp the details of our contribution is provided in:

- Chapter 2 elucidates the context of DRM systems, particularly how they relate to content streaming.
- Chapter 4 offers the required insights into the EME recommendation usage and workflow, which are integral to comprehending this contribution.

In addition, readers might want to look at the following chapters to get more insight on the subject beforehand:

- Chapter 5 gives a comprehensive approach to the Widevine protocol, cryptographic key ladder, and primitives involved.
- Chapter 7 provides a first overview of the Widevine protocol used through the EME API.

Contents

8.1. Context and Motivation

As seen throughout this thesis, Widevine is extensively used across various platforms, thanks in part to its compatibility with the EME API of the World Wide Web Consortium (W3C) [Can23]. However, despite its widespread adoption, the formal verification of the Widevine protocol within the EME context remains an underexplored area. Indeed, the primary concern that motivates this research is the lack of formal security guarantees in DRM systems like Widevine, mainly when available to every website when used in conjunction with EME. While these systems are designed to protect multimedia assets, the absence of formal verification leaves them vulnerable to potential security risks. Moreover, since EME only represents entry points to the underlying system, the complexity and proprietary nature of Widevine make it challenging to assess its security rigorously. This research aims to bridge this gap by employing formal methods to verify security properties that DRM systems should cover.

Research Gap and Importance. The existing literature primarily focuses on the practical aspects of DRM systems, such as efficiency and compatibility, often neglecting the formal verification of their security properties. This oversight can lead to a false sense of security and potential vulnerabilities. Given the critical role of DRM systems in protecting valuable digital assets, it is imperative to establish formal security guarantees. To achieve our objectives, we employ the Tamarin prover [SMCB12; MSCB13] to model and analyze the Widevine protocol as exposed by EME. This tool allows us to define security goals and rules formally, providing a rigorous framework for verification. However, it is worth noting that this work is ongoing, and definitive conclusions are yet to be drawn.

Contributions. We summarize our contributions as follows:

- We provide a full overview of the Widevine protocol under the EME API.
- We propose security goals for DRM systems regarding asset protection.
- We present a representation of the Widevine protocol used by EME using the tool Tamarin to prove the defined security goals.

This research serves as a step towards a more secure and reliable DRM ecosystem. By formally verifying the Widevine protocol, we aim to contribute to the broader understanding of DRM systems security and reduce companies and users reliance on unverified systems.

8.2. Widevine EME Protocol Internals

This section presents the full internal workings of the Widevine EME protocol. The protocol involves multiple actors, including the Content Decryption Module (CDM), the web browser through the EME API, as well as license and Over-the-Top (OTT) servers. Each actor plays a distinct role in the secure delivery, decryption, and playback of protected media content. The following subsections will elaborate on the roles of these actors, the initial state requirements, and the processes of license acquisition and renewal. The overall flow of the protocol is illustrated in Figure 8.1.

Figure 8.1. – EME protocol with Widevine specific message contents.

8.2.1. Main Actors

In this section, we outline the key actors involved in the Widevine EME protocol, detailing their responsibilities and interactions in the DRM ecosystem.

Widevine CDM. The Widevine CDM is responsible for handling license requests, decryption of content keys, decrypting protected media, and enforcing license policies. It is provisioned with a Client ID, a Device RSA Key, and a service certificate public key from the remote license server.

Web Browser (EME API). With the EME API, the web browser serves as the interface between distant servers and the CDM. It forwards license requests and responses between the CDM and the license server while abstracting implementation-specific details of the DRM system.

License & OTT servers. The license server is responsible for processing license requests from the CDM, verifying signatures, and issuing licenses. It maintains a database of keys indexed by Key IDs (KID) and generates session-specific keys for license provisioning as well as the service certificate for privacy mechanism. The OTT server provides the necessary metadata for initiating sessions and is the source of the media content to be decrypted and played.

8.2.2. Detailed Workflow of Widevine EME Protocol Operations

> **Reminder:** As seen in section 5.4, the Widevine controlflow involves various cryptographic keys, intermediate Root-of-Trust (RoT) such as the Device RSA Key, session-derived ones such as the Asset Key, MAC Client and Server Keys for authentication checks and the media Content Keys.

In this section, we present the specific stages and operations of the Widevine EME protocol, from the initial state setup to license acquisition and renewal processes.

Initial State. As elaborated in Chapter 5 and Chapter 7, the Widevine CDM is provisioned with a Client ID and a Device RSA Key. Additionally, it possesses a service certificate public key obtained from the remote license server, which holds the corresponding private key. These elements form the foundational mechanisms for the Widevine protocol and allow the CDM to proceed with license acquisition. Once the necessary information has been retrieved from the OTT server, a session within the Widevine CDM can be initiated, setting the stage for license acquisition.

8.2.2.1. License Acquisition

Client Request. Initially, the CDM creates a nonce in its memory map, which is incorporated into the license request to serve as an anti-replay measure. Using the public key of the service certificate from the OTT, the CDM secures a freshly generated privacy key. This privacy key is employed to encrypt the device Client ID, to avoid privacy concerns (details in Chapter 7). Alongside these elements, the request also includes a newly generated Request ID, the desired KID extracted from the media metadata, and the Request Time

T_1 which is stored in the CDM memory. The assembled request is signed using the Device RSA private key of the device and then sent to the license server to obtain the content key.

Server Response. Upon receiving the license request, the license server decrypts the Client ID and verifies the signature. It then generates a Session Key, which is used in conjunction with the request buffer to derive multiple HMAC keys: the MAC Server Key and the MAC Client Key, as well as an encryption key known as the Asset Key. The server retrieves the requested Content Key by searching for the KID in its license database and encrypts this key using the Asset Key. The generated MAC keys are stored in the server memory and indexed by a newly generated OTT session ID, which is included in the response. Also included in the response are the previous request ID, request time T_1, license policies regulating content key usage, KID, the encrypted Content Key, and its Key Control Block (KCB) which serves as metadata for the content key and includes the nonce, usage rights, and Time-To-Live (TTL). The response is then secured with an HMAC using the MAC Server Key and sent back to the CDM, along with the Session Key encrypted using the Device RSA public key of the client device.

Key Processing. Upon receiving the license response, the CDM decrypts the Session Key using its Device RSA private key and derives the same set of keys: MAC Server Key, MAC Client Key, and Asset Key. The CDM then verifies the HMAC of the received response and compares the stored T_1 with the one received in the response. After successful verification, it decrypts the Content Key and saves it with its KCB and reception time, and stores both the license policies, which include a license duration time Δ_1, and the OTT session ID in its memory. The nonce previously generated for the request is also removed from the CDM memory. During usage, if the client current time exceeds T_1 plus Δ_1 or the TTL from the KCB plus the reception time, the license is considered to have expired.

8.2.2.2. License Renewal

Client Request. If stored license policies allow the CDM to update its Content Key on usage or at a given time, a renewal request can be sent to the license server. On renewal request creation, a newly generated privacy key and IV are used to encrypt the Client ID to be sent. The privacy key is then protected by the service certificate as in the original request. Alongside them, the initial request ID and request time T_1 from the license request are used with a renewal request time T_2, a counter C set to 0, or to the current counter in memory if it is not the first renewal request, and the OTT session ID. The whole request is then protected by an HMAC using the MAC Client Key and sent to the license server.

Server Response. Upon receiving the request, the license server validates the HMAC and fetches the necessary license policies using the OTT session ID. The server then constructs a response that includes the original request ID, OTT session ID, T_1 and T_2, an incremented counter $C = C+1$, and the updated policies with the new license duration time Δ_2.

Key Update. The CDM can now update its Content Key. If the HMAC validation is successful, the new counter value replaces the existing one in memory, only if it is strictly greater than the former value; otherwise, the key update is unsuccessful. The internal function `RefreshKeys()` extends the lifespan of the content key as follows: $lifetime = \Delta_2 + T_2 - T_2'$. In

this equation, T_2' denotes the time obtained from the license response. Since multiple renewal requests with varying T_2 values may have been sent, the received T_2 might not correspond to the most recent one stored in the CDM. Once again, if the client current time now exceeds T_2 plus Δ_2, the license is considered to have expired.

Mobile Specific Within the Android implementation of Widevine EME, the CDM incorporates differences during the license renewal process. On request, a nonce is generated by the client and sent in the protected renewal request to the license server. This server respond as previously while adding in the response a new KCB with updated TTL to be replaced in the CDM after nonce verification and removal.

8.3. Security Goals

In this section, we recall the threat model considered when dealing with DRM systems and define its capabilities. We then present the security goals considered for DRM within our formal verification.

8.3.1. Threat Model

In the context of DRM systems like Widevine, it is not uncommon for the attacker to be a legitimate user with full control over their device. Such an attacker is well-versed in the intricacies of the Widevine protocol, its functions, and its behavior. Unlike traditional threat models that consider external attackers, our model focuses on this sophisticated user who aims to exploit the system from within. The attacker has dual avenues for exploitation:

EME JavaScript API Access. The attacker can interact with the protocol at the browser level using the EME API, potentially manipulating license requests and responses, and invoking calls to `MediaKey` and `MediaKeySession` functions.

Widevine API Access. The attacker can directly communicate with the underlying Widevine API, allowing them to use its cryptographic primitives as oracles, and call direct functions in order to forge proprietary messages like `GenerateRSASignature()`, `GenerateSignature()`, `DeriveKeysFromSessionKey()`, or `GenerateNonce()`.

8.3.2. High-level Goals

OTT providers rely on DRM systems to protect media from piracy and any unauthorized access to content. To this end, DRM prevent wild copy and control media distribution as well as their use duration by enforcing the three following principles:

- Prevent unauthorized copy of media.
- Controlled media distribution.
- Controlled consumption time.

Below, we defined six high-level goals based on these principles with their technical implication for Widevine EME that should be verified to ensure content protection.

Goal 1 (G1). *The OTT prevents clients from copying the media being displayed. Only the OTT and a valid CDM can decrypt the media content.* For Widevine, a valid CDM refers to a device with unrevoked certificates.

Goal 2 (G2). *The OTT limits the number of time a DRM-protected license can be used. A license response cannot be modified, and it can be loaded only by one CDM and only once by this CDM.* For Widevine, load refers to the CDM operation `LoadKeys()`.

Goal 3 (G3). *The OTT knows the clients to whom it issues a license. The license response can be loaded only on the CDM from which the related request comes.* For Widevine, it is the device signing the related license request with its Device RSA Key through `GenerateRSASignature()`.

Goal 4 (G4). *The OTT limits the number of time a DRM-protected renewal response can be used. A license renewal response cannot be modified, and it can be loaded only by one CDM and only once by this CDM.* For Widevine, it is the device hashing the request with its MAC Client Key.

Goal 5 (G5). *The OTT knows the clients to whom it issues a license renewal response. The license renewal response can be loaded only on the CDM from which the related request comes.* For Widevine, it is the device hashing the related license renewal request with its MAC CLient Key using `GenerateSignature()`. This key is derived through `DeriveKeysFromSessionKey()` from the original license request to which the associated license response was generated .

Goal 6 (G6). *A license lifetime can be extended if and only if its current license enforcement rule allows it.* For Widevine, a license cannot be renewed if the related policy currently stored is nonrenewable.

8.4. Formal Security Analysis

With the protocol and security goals defined in the previous sections 8.2 and 8.3, we present a first approach to implement the Widevine EME protocol in Tamarin, a formal prover based on the symbolic model.

8.4.1. Tamarin Prover

The Tamarin prover [SMCB12; MSCB13], is an advanced, automated tool designed for the verification of security protocols within the symbolic model framework. Tamarin has demonstrated its capabilities by identifying vulnerabilities and endorsing modifications in significant real-world protocols, such as the 5G AKA protocol [BDH+18; CD19], TLS 1.3 [CHH+17], the Noise framework [GHS+20], and the EMV payment card protocols [BST21b; BST21a]. Protocols are characterized using MultiSet Rewriting systems (MSR). Each rewrite rule in this context symbolizes a step or action undertaken by either a protocol participant or the adversary.

8.4.2. Example of Security Goal Verification using Tamarin

In this section, we present our Tamarin model, which aims to formally verify the security goals outlined earlier. The model is designed to capture the complex interactions between the OTT service, the Widevine CDM, and the attacker, who is assumed to be a sophisticated user with full control over their device and a deep understanding of the Widevine protocol.

Listing 8.1 provides the initial state of the system, focusing on the generation of keys. In this listing, rules like `GenOTTKey` and `GenDeviceRSAKey` generate fresh keys for the OTT service and the device, respectively. The *Out* in these rules represents the public channel where the public part of these keys is broadcasted, known by everyone, and available for future rules to be applied to them. For instance, `GenOTTKey` generates a fresh OTT key $kOTT$ and exports its public key to the public channel using *Out(pk(kOTT))*.

Listing 8.2 describes the initialization of a device with a Widevine CDM. The rule `CDMInit` generates a *clientID* that includes the public part of the Device RSA Key and its signature chain, among other elements. This *clientID* is then associated with a fresh *deviceID* to represent a specific device.

The Tamarin model is verified against the security goals using automated proof techniques. Listing 8.3 focuses on our security Goal 3, which aims to ensure that license responses are loaded only by the device that generated the corresponding request. This is crucial for maintaining the integrity of the DRM system and preventing unauthorized access to content.

- `CDMKeysInit` ensures that a device has derived its keys from a session key, confirming the presence of a session ID and the necessary cryptographic keys.
- `UniqueDeviceRSAKey` imposes that two distinct devices cannot share the same Device RSA key pair, highlighting potential issues in desktop devices as discussed in Chapter 7.
- `GenerateRequestBeforeLoad` aims to prove that the device loading a license response is the one that generated the corresponding request, thereby ensuring the integrity of the license exchange process.

```
rule GenOTTKey: // ServiceCertificate keys
  [ Fr(~kOTT)]
--[ LOTTKey(~kOTT) ]->
  [!OTTKey($ottID, ~kOTT),
    Out(pk(~kOTT)) ]

rule GenCertificate: // Root Widevine Cert
  [ Fr(~rootcert)]
--[ LCertificate(~rootcert) ]->
  [ !Certificate(~rootcert),
    Out(pk(~rootcert)) ]

rule GenDeviceSignKey: // Intermediate Cert
  [ Fr(~kSign),
    !Certificate(~rootcert) ]
 --[ LDeviceSignKey(~kSign) ]->
  [ !DeviceSignKey(~kSign),
    Out(pk(~kSign)),
    Out(sign(pk(~kSign), ~rootcert)) ]

rule GenDeviceRSAKey: // Device RSA Key
  [ Fr(~kDevice),
    !DeviceSignKey(~kSign) ]
--[ LDeviceRSAKey(~kDevice) ]->
  [ !DeviceRSAKey(~kDevice),
    Out(pk(~kDevice)),
    Out(sign(pk(~kDevice), ~kSign)) ]

rule GenMovie: // movie
  [ Fr(~movie) ] --> [ !MovieGen(~movie) ]

/* Create a new key for an existing movie */
rule GenMovieKeys:
  [ Fr(~kContent),
    !MovieGen(~movie) ]
--[ LMovieKey($title, ~movie, $ottID, $keyID, ~kContent) ]->
  [ !Movie($title, ~movie, $ottID, $keyID, ~kContent) ]
```

Listing 8.1 – Tamarin Code Key Initialization

```
/* Initialise a CDM: creates a persistent
   CDM storing the device rsa key kDevice and clientID*/
rule CDMInit:
 let clientID =
   <$serialNumber,
    pk(~kDevice),
    sign(pk(~kDevice), ~kSign),
    pk(~kSign),
    sign(pk(~kSign), ~rootCert),
    $clientInfo>
 in
  [ Fr(~deviceID),
    !Certificate(~rootCert),
    !DeviceRSAKey(~kDevice),
    !DeviceSignKey(~kSign) ]
--[ CDMInitL(~deviceID, ~kDevice, clientID) ]->
  [ !CDMInit(~deviceID, ~kDevice, clientID) ]
```

Listing 8.2 – Tamarin Code for CDM Initial State.

```
lemma CDMKeysInit[reuse, use_induction]:
"All #i deviceID sessionID kEnc kMacS kMacC.
    CDMKeys(sessionID, deviceID, kEnc, kMacS, kMacC)@i ==>
    ((Ex #j.
    #j < #i & CDMDeriveKeys(sessionID, deviceID, kEnc, kMacS, kMacC)@j) |
    (kEnc ='null'))"

restriction UniqueDeviceRSAKey:
"All #i #j deviceID1 kDevice clientID1 deviceID2 clientID2.
    CDMInitL(deviceID1, kDevice, clientID1)@i &
    CDMInitL(deviceID2, kDevice, clientID2)@j ==>
    #i = #j"

lemma GenerateRequestBeforeLoad:
"All #i1 #i2 deviceID kDevice clientID sessionID
                kContent kEnc kMacS kMacC receivedRequestID
                licencePolicy keyID1 keyID2 iv1 enc_kContent
                ttl usage n title movie ottID.
    CDMLoad(sessionID, deviceID, kDevice, clientID,
            kContent, kEnc, kMacS, kMacC,
            <receivedRequestID, licencePolicy, keyID1,
            iv1, enc_kContent, ttl, usage, n>)@i1 &
    LMovieKey(title, movie, ottID, keyID2, kContent)@i2 ==>
    Ex #j clientID2 sessionID2 keyID3 enc_clientID enc_privacyKey iv2.
    (CDMRSASignature(sessionID2, deviceID, kDevice, clientID2,
                        <keyID3, receivedRequestID,
                        enc_clientID, iv2, enc_privacyKey>)@j
        & #j < #i1)"
```

Listing 8.3 – Tamarin Code for Goal 3.

8.5. Discussion and Current Limitations

This chapter serves as a first effort to formally verify the security properties of the Widevine DRM protocol, particularly within the EME context. Our focus on Widevine, although justified by its widespread adoption, leaves out other DRM solutions like Microsoft PlayReady or Apple FairPlay Streaming. Given the proprietary nature of these systems, extending our formal methods to these platforms remains a challenge and an avenue for future research. In addition, while the Tamarin prover has been instrumental in our formal verification process, allowing us to define and test security goals rigorously, our work is ongoing, and our implementation still needs to be improved and tested on the whole protocol.

Part III.

Conclusion and Future Work

Conclusion

It's so much easier to see the world in black and white. Gray? I don't know what to do with gray...

Garrus – Mass Effect

Through a series of empirical analyses and technical investigations, this thesis has demystified various dimensions of Widevine opaque nature, further informed by our developed artifacts. In the subsequent sections, we elucidate our answers to the research questions initially posed.

Widevine Security Aspect. We first explored vulnerabilities inherent to Widevine opaque nature, emphasizing their implications for the overall security of protected media content. Motivated by the extensive utilization of Widevine in numerous Over-the-Top (OTT) platforms, this study uncovers new vulnerabilities that directly affect the Widevine security properties.

We demonstrated the practical impact of each discovered vulnerability through the development of a full practical Proof-of-Concept (PoC), aiming to bring serious consideration to these issues within the computer security community and among stakeholders. Our tools WideXtractor and PoC L3 key recovery substantiate these findings, with the latter allowing for full media recovery. Specifically, the investigation leads to the following insights:

- Android OEM Crypto library implementation had notable security gaps, which were disclosed and subsequently patched, indicating an environment with potential for exploitation. The acknowledgment from Google and Netflix Hall of Fame underscores the significance of these vulnerabilities for both OTT and Digital Rights Management (DRM) developers.
- While Widevine opacity aims to secure the content, it inadvertently gives rise to an insecure-by-default environment, mirroring issues commonly observed in other closed-source systems.
- The opaque nature of Widevine L3 and its complicated relationship with OTT platforms further amplifies these security flaws, contributing to the misusage of the solution and thus, weakening the protection of media assets.

We made both our tool WideXtractor and our PoC publicly available, aiming to arm the community with resources to better understand and safeguard against such vulnerabilities. Although these vulnerabilities were patched, the findings underscore the limitations of proactive security measures when dealing with opaque systems.

Privacy Concerns within DRMs. In addition to investigating the security aspects of Widevine, we looked into the privacy implications arising from its opaque and closed-source nature. We chose to focus on this area due to current concerns around online tracking and the collection of user data by various web services.

To demonstrate the practical consequences of these privacy concerns, we developed a full practical PoC through our tool EME Tracker. This tool illuminates how websites can exploit the Encrypted Media Extension (EME) API to collect unique identifiers from users device, thus enabling online tracking. The key observations drawn from this section are as follows:

- Web browsers misimplementation of privacy features, influenced by the opacity of Widevine, results in significant privacy threats, substantiated by the acknowledgment from Mozilla Security Bug Bounty Program Hall of Fame.
- Despite intended measures to preserve user privacy within the EME API, our analysis exposes loopholes that can be exploited for online tracking. This underlines the importance of transparent and open-source implementations for safeguarding user privacy.
- The opacity of Widevine inadvertently complicates the task of preserving privacy, leading to unintended attack vectors for user tracking.

Our PoC, EME Tracker, is publicly accessible, aiming to draw attention to the importance of transparency and accuracy in the implementation of privacy features in DRM systems. Even though the identified issues have been patched in Mozilla Firefox, the results call attention to the existence of such systems exposed through web APIs.

Our research questions. We structured this thesis around three main research questions, as initially laid out in section 1.1. Here, we summarize the responses to these questions, incorporating the results from our investigations:

I. *What is the impact of the opaque nature of Widevine regarding vulnerability discovery, and what are the repercussions on the overall security of protected media content?*

We revealed a significant security flaw in Widevine architecture that enables full media recovery, as discussed in Chapter 5. This finding emphasizes the double-edged sword of opacity: while it might serve to keep certain functionalities secret, it equally cloaks potential vulnerabilities, making them harder to identify and rectify. Our contributions, especially the tool WideXtractor, have led to patches that mend these security gaps, highlighting the real-world impacts of our work. Chapter 8 tries to go deeper in this investigation by providing a formal security analysis using our current understanding of the protocol.

II. *How is Widevine leveraged by OTT platforms considering its opaque implementations?*

Our research in Chapter 6 shows that OTT platforms utilization of Widevine often falls short of its full capabilities. The absence of clear implementation guidelines contributes to less effective usage, compromising both content protection and system efficacy.

III. *In what ways does the proprietary and closed-source Widevine protocol influence privacy concerns for end users, particularly in terms of online tracking?*

Our exploration in Chapter 7 reveals that Widevine implementation choices inadvertently facilitate tracking and privacy infringement. Our tool, EME Tracker, proved instrumental in identifying these issues, leading to patches that mitigate the vulnerabilities. Our findings urge a reconsideration of how privacy features are implemented and understood in such proprietary systems.

Perspectives and Current Work

This chapter gives an overview of ongoing and future works to pursue research started by this thesis.

Tool-Assisted DRM Vulnerability Discovery

Given the newfound insights into the inner workings of Widevine and the vulnerabilities uncovered, the path forward offers an exciting opportunity to leverage automated tools to accelerate vulnerability research. Utilizing fuzzers such as AFL++ [FMEH20], which can systematically probe the system for weaknesses by generating and testing a vast array of input combinations, could significantly speed up the discovery process. With a more profound understanding of Widevine structure, these tools can be tailored to target specific areas of potential weakness such as cryptographic operations and license management. By integrating automated vulnerability discovery with manual inspection and analysis, future research could contribute to reshaping the landscape of digital rights management and the security paradigms surrounding opaque systems.

Microsoft PlayReady and Apple FairPlay Analysis

Building upon the comprehensive examination of the Widevine DRM, future research endeavors may extend our work to other significant players in the DRM landscape, namely Microsoft PlayReady [Mic23] and Apple FairPlay [App23].

Security. By applying the methodologies and insights gleaned from the investigation into Widevine, a parallel analysis could uncover how the opaque nature of these systems might reveal similar vulnerabilities or implementation challenges. This comparative approach offers the potential to discern common patterns and discrepancies among different DRM systems, deepening the understanding of how opacity influences security and privacy across diverse platforms. Such an expansive study could contribute valuable knowledge to the broader discourse on DRM, guiding both the technological development and policy considerations to foster more transparent, secure, and user-centric digital rights management.

Privacy. Expanding the analysis to Microsoft PlayReady and Apple FairPlay can illuminate how these DRM systems handle user data and privacy. Given the findings regarding privacy concerns in Widevine, similar vulnerabilities or problematic practices might be prevalent in these alternative systems as well. In addition, a cross-platform study could identify commonalities and differences in how user privacy is either compromised or protected within

each system. Evaluating metrics such as data collection, retention, and sharing policies alongside the cryptographic measures in place to safeguard this data will allow for a nuanced understanding of privacy in DRM ecosystems. This would significantly contribute to the scholarly discussion, providing empirical data that could influence future privacy regulations and DRM design philosophies.

Bibliography

[Ado] Adobe. *Primetime Digital Rights Management (DRM)*. https://experienceleague.adobe.com/docs/primetime/drm/home.html (cit. on pp. viii, 4).

[Adv] Advanced Access Content System Licensing Administrator. *AACS Website*. https://aacsla.com/ (cit. on pp. viii, 4, 14).

[Adv07] Advanced Access Content System Licensing Administrator. *Illegal Offering of Processing Key to Circumvent AACS Copyright Protection*. https://www.lumendatabase.org/notices/21725. 2007 (cit. on p. 14).

[AEE+14] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juárez, Arvind Narayanan, and Claudia Díaz. *The Web Never Forgets: Persistent Tracking Mechanisms in the Wild*. In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*. Ed. by Gail-Joon Ahn, Moti Yung, and Ninghui Li. ACM, 2014, pp. 674–689. URL: https://doi.org/10.1145/2660267.2660347 (cit. on p. 69).

[Anda] Android. *DRM Framework*. https://source.android.com/devices/drm (cit. on p. 22).

[Andb] Android. *ExoPlayer Documentation*. https://developer.android.com/reference/androidx/media3/exoplayer/ExoPlayer (cit. on p. 22).

[Andc] Android. *Introduction to Jetpack Media3*. https://developer.android.com/guide/topics/media/media3 (cit. on p. 22).

[Andd] Android. *Media Framework Hardening*. https://source.android.com/devices/media/framework-hardening (cit. on p. 22).

[Ande] Android. *MediaCrypto Documentation*. https://developer.android.com/reference/android/media/MediaCrypto (cit. on p. 22).

[Andf] Android. *MediaDrm Documentation*. https://developer.android.com/reference/android/media/MediaDrm (cit. on p. 22).

[And22] Android. *Protect against security threats with SafetyNet*. https://developer.android.com/training/safetynet. 2022 (cit. on pp. 47, 59).

[And23] Android. *Android Studio*. https://developer.android.com/studio/. 2023 (cit. on p. 82).

[App23] Apple. *Apple FairPlay*. https://developer.apple.com/streaming/fps/. 2023 (cit. on pp. viii, 4, 17, 67, 111).

[AWS+11] Mika D. Ayenson, Dietrich James Wambach, Ashkan Soltani, Nathaniel Good, and Chris Jay Hoofnagle. *Flash Cookies and Privacy II: Now with HTML5 and ETag Respawning*. In: 2011 (cit. on p. 69).

[BBC06] BBC News. *iTunes copy protection 'cracked'*. http://news.bbc.co.uk/2/hi/6083110.stm?lsm/. 2006 (cit. on p. 15).

[BBC07] BBC News. *Hi-def DVD security is bypassed*. http://news.bbc.co.uk/1/hi/technology/6301301.stm. 2007 (cit. on p. 14).

[BDH+18] David A. Basin, Jannik Dreier, Lucca Hirschi, Sasa Radomirovic, Ralf Sasse, and Vincent Stettler. *A Formal Analysis of 5G Authentication*. In: *CCS*. ACM, 2018, pp. 1383–1396 (cit. on p. 100).

[Ben16] Gal Beniamini. *QSEE privilege escalation vulnerability and exploit (CVE-2015-6639)*. https://bits-please.blogspot.com/2016/05/qsee-privilege-escalation-vulnerability.html. 2016 (cit. on pp. 35, 55).

[Ber17] Tim Berners-Lee. *On EME in HTML5*. https://www.w3.org/blog/2017/02/on-eme-in-html5. 2017 (cit. on p. 67).

[Blo22] Netflix Technology Blog. *Modernizing the Netflix TV UI Deployment Process*. https://netflixtechblog.medium.com/modernizing-the-netflix-tv-ui-deployment-process-28e022edaaef. 2022 (cit. on p. 89).

[Bra22a] Brave. *Brave Passes 50 Million Monthly Active Users, Growing 2x for the Fifth Year in a Row*. https://brave.com/2021-recap/. 2022 (cit. on p. 76).

[Bra22b] Brave Software, Inc. *Brave Browser*. https://brave.com/. 2022 (cit. on p. 70).

[BST21a] David A. Basin, Ralf Sasse, and Jorge Toro-Pozo. *Card Brand Mixup Attack: Bypassing the PIN in non-Visa Cards by Using Them for Visa Transactions*. In: *USENIX Security Symposium*. USENIX Association, 2021, pp. 179–194 (cit. on p. 100).

[BST21b] David A. Basin, Ralf Sasse, and Jorge Toro-Pozo. *The EMV Standard: Break, Fix, Verify*. In: *IEEE Symposium on Security and Privacy*. IEEE, 2021, pp. 1766–1781 (cit. on p. 100).

[Can23] Can I Use. *Encrypted Media Extensions*. https://caniuse.com/eme. 2023 (cit. on pp. 27, 49, 50, 57, 67, 80, 95).

[CD19] Cas Cremers and Martin Dehnel-Wild. *Component-Based Formal Analysis of 5G-AKA: Channel Assumptions and Session Confusion*. In: *NDSS*. The Internet Society, 2019 (cit. on p. 100).

[CHH+17] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. *A Comprehensive Symbolic Analysis of TLS 1.3*. In: *CCS*. ACM, 2017, pp. 1773–1788 (cit. on p. 100).

[CLB+22] Darion Cassel, Su-Chin Lin, Alessio Buraggina, William Wang, Andrew Zhang, Lujo Bauer, Hsu-Chun Hsiao, Limin Jia, and Timothy Libert. *"OmniCrawl: Comprehensive Measurement of Web Tracking With Real Desktop and Mobile Browsers"*. In: *Proc. Priv. Enhancing Technol.* 2022.1 (2022), pp. 227–252 (cit. on p. 82).

[Cou22a] Stat Counter. *Desktop Browser Market Share Worldwide*. https://gs.statcounter.com/browser-market-share/desktop/worldwide/2022. 2022 (cit. on p. 76).

[Cou22b] Stat Counter. *Mobile Browser Market Share Worldwide*. https://gs.statcounter.com/browser-market-share/mobile/worldwide/2022. 2022 (cit. on p. 76).

[CSFP20] David Cerdeira, Nuno Santos, Pedro Fonseca, and Sandro Pinto. *SoK: Understanding the Prevailing Security Vulnerabilities in TrustZone-assisted TEE Systems*. In: *IEEE Symposium on Security and Privacy*. IEEE, 2020, pp. 1416–1432 (cit. on p. 55).

[CWW+18] Sze Yiu Chau, Bincheng Wang, Jianxiong Wang, Omar Chowdhury, Aniket Kate, and Ninghui Li. *Why Johnny Can't Make Money With His Contents: Pitfalls of Designing and Implementing Content Delivery Apps*. In: *Proceedings of the 34th Annual Computer Security Applications Conference*. ACSAC '18. San Juan, PR, USA: Association for Computing Machinery, 2018, pp. 236–251. ISBN: 9781450365697. URL: https://doi.org/10.1145/3274694.3274752 (cit. on pp. 55, 60).

[Dav19] David Buchanan. *Breaking Widevine L3 on Linux Chrome browser*. https://twitter.com/david3141593/status/1080606827384131590. 2019 (cit. on p. 18).

[DKS21] Ahaan Dabholkar, Sourya Kakarla, and Dhiman Saha. *"Looney Tunes: Exposing the Lack of DRM Protection in Indian Music Streaming Services"*. In: *CoRR* abs/2103.16360 (2021). arXiv: 2103.16360. URL: https://arxiv.org/abs/2103.16360 (cit. on p. 57).

[DRM05] DRMDBG. *Download the latest version of drmdbg.exe*. https://web.archive.org/web/20071224094751/http://stream-recorder.com/forum/download-latest-version-drmdbg-exe-t1959.html. 2005 (cit. on p. 15).

[DSWB19] David Dorwin, Jerry Smith, Mark Watson, and Adrian Bateman. *Encrypted Media Extensions*. https://www.w3.org/TR/encrypted-media/. 2019 (cit. on pp. ix, 4, 15, 28, 35, 55, 67, 81).

[DVD] DVD Copy Control Association. *DVDCCA Website*. http://www.dvdcca.org/ (cit. on pp. viii, 4, 14).

[Eck10] Peter Eckersley. *How Unique Is Your Web Browser?* In: *Privacy Enhancing Technologies, 10th International Symposium, PETS 2010, Berlin, Germany, July 21-23, 2010. Proceedings*. Ed. by Mikhail J. Atallah and Nicholas J. Hopper. Vol. 6205. Lecture Notes in Computer Science. Springer, 2010, pp. 1–18. URL: https://doi.org/10.1007/978-3-642-14527-8%5C_1 (cit. on pp. 70, 84).

[Ele17] Electronic Frontier Foundation. *Reported Case List*. https://www.eff.org/files/2016/03/17/1201_reported_case_list_revised.xls. 2017 (cit. on pp. ix, 4).

[Exp] Export Administration Regulations. *Export Administration Regulations Website*. https://www.bis.doc.gov/index.php/regulations/export-administration-regulations-ear (cit. on p. 14).

[FMEH20] Andrea Fioraldi, Dominik Christian Maier, Heiko Eißfeldt, and Marc Heuse. *AFL++ : Combining Incremental Steps of Fuzzing Research*. In: *WOOT @ USENIX Security Symposium*. USENIX Association, 2020 (cit. on p. 111).

[For22a] Fortune Business Insights. *Over The Top Services Market to Reach USD 139.00 Billion in 2028; Emergence of Smart TVs by Various Companies to Bolster Growth*. https://www.globenewswire.com/news-release/2021/08/17/2281647/0/en/Over-The-Top-Services-Market-to-Reach-USD-139-00-Billion-in-2028-Emergence-of-Smart-TVs-by-Various-Companies-to-Bolster-Growth-states-Fortune-Business-Insights.html. 2022 (cit. on pp. vii, 3, 35, 55).

[For22b] Dash Industry Forum. *Content Protection*. https://dashif.org/identifiers/content_protection/. 2022 (cit. on p. 28).

[Fou14] Free Software Foundation. *FSF condemns partnership between Mozilla and Adobe to support Digital Restrictions Management*. https://www.fsf.org/news/fsf-condemns-partnership-between-mozilla-and-adobe-to-support-digital-restrictions-management. 2014 (cit. on p. 67).

[GGK+16] Christina Garman, Matthew Green, Gabriel Kaptchuk, Ian Miers, and Michael Rushanan. *Dancing on the Lip of the Volcano: Chosen Ciphertext Attacks on Apple iMessage*. In: *USENIX Security Symposium*. USENIX Association, 2016, pp. 655–672 (cit. on p. 44).

[GHS+20] Guillaume Girol, Lucca Hirschi, Ralf Sasse, Dennis Jackson, Cas Cremers, and David A. Basin. *A Spectral Analysis of Noise: A Comprehensive, Automated, Formal Analysis of Diffie-Hellman Protocols*. In: *USENIX Security Symposium*. USENIX Association, 2020, pp. 1857–1874 (cit. on p. 100).

[Git20] Github. *DMCA*. https://github.com/github/dmca/blob/master/2020/11/2020-11-09-Google.md. 2020 (cit. on p. 58).

[Giz18] Gizmodo. *Apple Declares War on 'Browser Fingerprinting,' the Sneaky Tactic That Tracks You in Incognito Mode*. https://gizmodo.com/apple-declares-war-on-browser-fingerprinting-the-sneak-1826549108. 2018 (cit. on p. 70).

[GLB18] Alejandro Gómez-Boix, Pierre Laperdrix, and Benoit Baudry. *Hiding in the Crowd: an Analysis of the Effectiveness of Browser Fingerprinting at Large Scale*. In: *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*. Ed. by Pierre-Antoine Champin, Fabien Gandon, Mounia Lalmas, and Panagiotis G. Ipeirotis. ACM, 2018, pp. 309–318. URL: https://doi.org/10.1145/3178876.3186097 (cit. on pp. 70, 84, 85).

[Goo] Google. *Factory Images for Nexus and Pixel Devices*. https://developers.google.com/android/images (cit. on p. 38).

[Goo22a] Google. *EME Call and Event Logger*. https://chrome.google.com/webstore/detail/eme-call-and-event-logger/cniohcjecdcdhgmlofniddfoeokbpbpb. 2022 (cit. on p. 49).

[Goo22b] Google. *ExoPlayer*. https://github.com/google/ExoPlayer. 2022 (cit. on pp. 22, 58, 62).

[Goo22c] Google. *Learn when you'll get software updates on Google Pixel phones*. https://support.google.com/pixelphone/answer/4457705. 2022 (cit. on pp. 55, 62).

[Goo22d] Google. *The Privacy Sandbox*. https://privacysandbox.com/. 2022 (cit. on p. 70).

[Goo22e] Google. *User-Agent reduction*. https://developer.chrome.com/docs/privacy-sandbox/user-agent/. 2022 (cit. on p. 85).

[Goo22f] Google Widevine. *Widevine DRM Overview*. https://developers.google.com/widevine/drm/overview. 2022 (cit. on pp. 57, 61).

[Goo22g] Google Widevine. *Widevine News*. https://web.archive.org/web/20210903150435/https://www.widevine.com/news. 2022 (cit. on pp. 57, 58, 74, 79).

[Goo23a] Google. *Google Play Store: Firefox Focus: No Fuss Browser*. https://play.google.com/store/apps/details?id=org.mozilla.focus. 2023 (cit. on p. 77).

[Goo23b] Google. *Google Play Store: Ghostery Privacy Browser*. https://play.google.com/store/apps/details?id=com.ghostery.android.ghostery. 2023 (cit. on p. 77).

[Goo23c] Google. *Media3 ExoPlayer*. https://developer.android.com/guide/topics/media/exoplayer. 2023 (cit. on p. 22).

[Goo23d] Google Widevine. *Widevine*. https://widevine.com/. 2023 (cit. on pp. viii, 4, 17, 35, 55, 57, 58, 67).

[Gow22] Adam Gowdiak. *Microsoft PlayReady security research*. https://seclists.org/fulldisclosure/2022/Dec/10. 2022 (cit. on p. 17).

[Had21] Tomer Hadad. *Reversing the old Widevine Content Decryption Module*. https://github.com/tomer8007/widevine-l3-decryptor/wiki/Reversing-the-old-Widevine-Content-Decryption-Module. 2021 (cit. on pp. 18, 70).

[Hal17] Harry Halpin. *The Crisis of Standardizing DRM: The Case of W3C Encrypted Media Extensions*. In: *SPACE*. Vol. 10662. Lecture Notes in Computer Science. Springer, 2017, pp. 10–29 (cit. on pp. ix, 4, 35, 55, 88).

[Hég17] Philippe Le Hégaret. *Disposition of Comments for Encrypted Media Extensions and Director's decision*. https://lists.w3.org/Archives/Public/public-html-media/2017Jul/0000.html. 2017 (cit. on p. 67).

[Ins22] Fortune Business Insights. *Video Streaming Market Size*. https://www.fortunebusinessinsights.com/video-streaming-market-103057. 2022 (cit. on p. 67).

[Int13] Intertrust. *Baidu's iQIYI Licenses Intertrust's ExpressPlay Marlin DRM Solution*. https://web.archive.org/web/20130911143711/http://www.intertrust.com/news/press/iQIYI. 2013 (cit. on p. 16).

[Int22a] Intertrust. *ExpressPlay multi-DRM service DRM client compatibility*. https://go.intertrust.com/hubfs/assets/Media/ExpressPlay-multi-DRM-service-DRM-client-compatibility.pdf. 2022 (cit. on pp. 16, 18).

[Int22b] Intertrust. *Using Marlin DRM to protect non-audiovisual assets*. https://go.intertrust.com/hubfs/assets/Marlin/Using-Marlin-DRM-to-Protect-Non-Audiovisual-Assets.pdf. 2022 (cit. on p. 16).

[Ird] Irdeto. *Irdeto Forensic Watermarking*. https://irdeto.com/video-entertainment/forensic-watermarking/ (cit. on p. 18).

[ISO15] ISO/IEC. *Common encryption in ISO Base Media File Format files - 2nd Edition*. 2015 (cit. on pp. 35, 62).

[ISO16] ISO Central Secretary. *Information technology — MPEG systems technologies — Part 7: Common encryption in ISO base media file format files*. en. Standard ISO/IEC 23001-7:2016. International Organization for Standardization, 2016. URL: https://www.iso.org/standard/68042.html (cit. on p. 17).

[ISO22] ISO Central Secretary. *Information technology — Dynamic adaptive streaming over HTTP (DASH) — Part 1: Media presentation description and segment formats*. en. Standard ISO/IEC 23009-1:2022. International Organization for Standardization, 2022. URL: https://www.iso.org/standard/83314.html (cit. on p. 17).

[Jaf17] Jeff Jaffe. *Reflections on the EME debate*. https://www.w3.org/blog/2017/09/reflections-on-the-eme-debate. 2017 (cit. on p. 67).

[Jer07] Jeremy Reimer. *First pirated HD DVD movie hits BitTorrent*. https://arstechnica.com/uncategorized/2007/01/8622/. 2007 (cit. on p. 14).

[Kam10] Samy Kamkar. *evercookie*. https://samy.pl/evercookie/. 2010 (cit. on p. 69).

[KCLR07] Ton Kalker, Knox Carey, Jack Lacy, and Martin Rosner. *The Coral DRM Interoperability Framework*. In: *CCNC*. IEEE, 2007, pp. 930–934 (cit. on p. 16).

[kka22] kkapsner. *CanvasBlocker*. https://addons.mozilla.org/en-US/firefox/addon/canvasblocker/. 2022 (cit. on p. 70).

[Lau22] Laurent Piron. *DASH-IF IOP Watermarking*. Standard. DASH Industry Forum, 2022 (cit. on p. 18).

[lem00] lemuria.org. *DeCSS*. https://www.lemuria.org/DeCSS/decss.html. 2000 (cit. on p. 14).

[LISP22] Xu Lin, Panagiotis Ilia, Saumya Solanki, and Jason Polakis. *Phish in Sheep's Clothing: Exploring the Authentication Pitfalls of Browser Fingerprinting*. In: *USENIX Security Symposium*. USENIX Association, 2022, pp. 1651–1668 (cit. on p. 85).

[LLM21] Wu Liao, Luo Luo, and Zhu Mi. *Research on Fairplay DRM and Obfuscation Implementation*. https://segmentfault.com/a/1190000041023774. 2021 (cit. on p. 17).

[LRB16] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. *Beauty and the Beast: Diverting Modern Web Browsers to Build Unique Browser Fingerprints*. In: *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*. IEEE Computer Society, 2016, pp. 878–894. URL: https://doi.org/10.1109/SP.2016.57 (cit. on pp. 70, 84).

[Mao22] Giorgio Maone. *NoScript Official Website*. https://noscript.net/. 2022 (cit. on p. 70).

[Mara] Marlin Community. *Marlin DRM technology*. https://www.marlin-community.com/ (cit. on pp. viii, 4, 16).

[Marb] Marlin Trust. *MTMO Website*. https://marlin-trust.com/ (cit. on p. 16).

[Mar06] Mark Hachman. *Microsoft To Issue Fix For DRM Stripper App*. https://web.archive.org/web/20100619061323/https://www.pcmag.com/article2/0,2817,2009515,00.asp. 2006 (cit. on p. 15).

[Mar10] Marlin Community. *Marlin Broadband Specifications*. Standard. 2010 (cit. on p. 16).

[Mar14] Marlin Community. *Marlin DRM eBook Extension specifications*. Standard. 2014 (cit. on p. 16).

[Mar17] Marlin Community. *Marlin Simple Secure Streaming Specifications*. Standard. 2017 (cit. on p. 16).

[May09] Jonathan R Mayer. *"Any person... a pamphleteer: Internet Anonymity in the Age of Web 2.0"*. In: *Undergraduate Senior Thesis, Princeton University* 85 (2009) (cit. on p. 70).

[MBYS11] Keaton Mowery, Dillon Bogenreif, Scott Yilek, and Hovav Shacham. *Fingerprinting Information in JavaScript Implementations*. In: *Proceedings of W2SP 2011*. Ed. by Helen Wang. IEEE Computer Society. May 2011 (cit. on p. 70).

[McC02] Declan McCullagh. *Security warning draws DMCA threat*. https://www.cnet.com/news/security-warning-draws-dmca-threat/. 2002 (cit. on pp. ix, 4).

[Mdn23] Mdn Web Docs. *User-Agent*. https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/User-Agent. 2023 (cit. on p. 85).

[MGS+17] Aravind Machiry, Eric Gustafson, Chad Spensky, Christopher Salls, Nick Stephens, Ruoyu Wang, Antonio Bianchi, Yung Ryn Choe, Christopher Kruegel, and Giovanni Vigna. *BOOMERANG: Exploiting the Semantic Gap in Trusted Execution Environments*. In: *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society, 2017. URL: https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/boomerang-exploiting-semantic-gap-trusted-execution-environments/ (cit. on p. 55).

[Mica] Microsoft. *PlayReady Security Level*. https://learn.microsoft.com/en-us/playready/overview/security-level (cit. on p. 17).

[Micb] Microsoft. *Windows Media DRM Client Extended APIs*. hhttps://learn.microsoft.com/en-us/windows/win32/wmformat/windows-media-drm-client-extended-apis (cit. on pp. viii, 4).

[Mic23] Microsoft. *Microsoft PlayReady*. https://www.microsoft.com/playready/. 2023 (cit. on pp. viii, 4, 16, 67, 111).

[MIT22] MITRE. *CVE List*. https://cve.mitre.org. 2022 (cit. on pp. ix, 5).

[Mov18] MovieLabs. *MovieLabs Specification for Enhanced Content Protection - Version 1.2*. Standard. 2018 (cit. on p. 16).

[Moz23] Mozilla. *Enhanced Tracking Protection in Firefox for desktop*. https://support.mozilla.org/en-US/kb/enhanced-tracking-protection-firefox-desktop. 2023 (cit. on p. 70).

[MRH+13] Martin Mulazzani, Philipp Reschl, Markus Huber, Manuel Leithner, Sebastian Schrittwieser, Edgar Weippl, and FC Wien. *Fast and reliable browser identification with javascript engine fingerprinting*. In: *Web 2.0 Workshop on Security and Privacy (W2SP)*. Vol. 5. Citeseer. 2013, p. 4 (cit. on p. 70).

[MS12] Keaton Mowery and Hovav Shacham. *Pixel Perfect: Fingerprinting Canvas in HTML5*. In: *Proceedings of W2SP 2012*. Ed. by Matt Fredrikson. IEEE Computer Society. May 2012 (cit. on p. 70).

[MSCB13] Simon Meier, Benedikt Schmidt, Cas Cremers, and David A. Basin. *The TAMARIN Prover for the Symbolic Analysis of Security Protocols*. In: *CAV*. Vol. 8044. Lecture Notes in Computer Science. Springer, 2013, pp. 696–701 (cit. on pp. xii, 7, 95, 100).

[MT88] Ryoichi Mori and Shuichi Tashiro. *"The Concept of Software Service System (SSS)"*. In: *Systems and Computers in Japan* 19.5 (1988), pp. 38–49 (cit. on pp. viii, 3, 14).

[Mul22] Multilogin. *Canvas Defender*. https://addons.mozilla.org/en-US/firefox/addon/no-canvas-fingerprinting/. 2022 (cit. on p. 70).

[NAG] NAGRA KUDELSKI. *NAGRA Website*. https://dtv.nagra.com/ (cit. on p. 18).

[Nat] National Security Agency. *Ghidra SRE*. https://ghidra-sre.org/ (cit. on p. 38).

[Net20] Netflix. *Message Security Layer*. https://github.com/Netflix/msl. 2020 (cit. on pp. 48, 89).

[nic23] nicolo.dev. *Analysis of Obfuscations Found in Apple FairPlay*. https://nicolo.dev/blog/fairplay-apple-offuscamento/. 2023 (cit. on p. 17).

[OMA] OMA SPecWorks. *OMA SpecWorks Website*. https://omaspecworks.org/ (cit. on p. 15).

[Ope02] Open Mobile Alliance. *Digital Rights Management Version 1.0*. Standard. OMA Open Mobile Alliance, 2002 (cit. on p. 15).

[Ope06] Open Mobile Alliance. *Secure Content Exchange Requirements*. Standard. OMA Open Mobile Alliance, 2006 (cit. on p. 15).

[Ope08] Open Mobile Alliance. *DRM Specification 2.0*. Standard. OMA Open Mobile Alliance, 2008 (cit. on p. 15).

[Ope09a] Open Mobile Alliance. *OMA Secure Removable Media Requirements V1.0*. Standard. OMA Open Mobile Alliance, 2009 (cit. on p. 15).

[Ope09b] Open Mobile Alliance. *OMA Secure Removable Media Requirements V1.1*. Standard. OMA Open Mobile Alliance, 2009 (cit. on p. 15).

[Pan] Pancake. *Radare2*. https://rada.re/n/ (cit. on p. 38).

[Por22] PortSwigger. *Burp Suite*. https://portswigger.net/burp. 2022 (cit. on p. 60).

[Pro22] Tor Project. *Tor Browser*. https://www.torproject.org/. 2022 (cit. on p. 70).

[PSF22a] Gwendal Patat, Mohamed Sabt, and Pierre-Alain Fouque. *Exploring Widevine for Fun and Profit*. In: *43rd IEEE Security and Privacy, SP Workshops 2022, San Francisco, CA, USA, May 22-26, 2022*. IEEE, 2022, pp. 277–288. URL: https://doi.org/10.1109/SPW54247.2022.9833867 (cit. on pp. xi, xv, 6, 70, 71, 74, 83).

[PSF22b] Gwendal Patat, Mohamed Sabt, and Pierre-Alain Fouque. *WideLeak: How Over-the-Top Platforms Fail in Android*. In: *52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2022, Baltimore, MD, USA, June 27-30, 2022*. IEEE, 2022, pp. 501–508. URL: https://doi.org/10.1109/DSN53405.2022.00056 (cit. on pp. xi, xv, 6, 70, 82).

[PSF23] Gwendal Patat, Mohamed Sabt, and Pierre-Alain Fouque. *"Your DRM Can Watch You Too: Exploring the Privacy Implications of Browsers (mis)Implementations of Widevine EME"*. In: *Proc. Priv. Enhancing Technol.* 2023.4 (2023), pp. 306–321 (cit. on pp. xii, xv, 7).

[Rav] Ole André V. Ravnås. *Frida*. https://frida.re/ (cit. on pp. 40, 59).

[rla22] rlaphoenix. *pywidevine*. https://github.com/rlaphoenix/pywidevine. 2022 (cit. on p. 70).

[SAB15] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. *Trusted Execution Environment: What It is, and What It is Not*. In: *TrustCom/BigDataSE/ISPA (1)*. IEEE, 2015, pp. 57–64 (cit. on p. 55).

[Sch01] John Schwartz. *Giving Web a Memory Cost Its Users Privacy*. https://www.nytimes.com/2001/09/04/business/giving-web-a-memory-cost-its-users-privacy.html. 2001 (cit. on p. 69).

[SCM+10] Ashkan Soltani, Shannon Canty, Quentin Mayo, Lauren Thomas, and Chris Jay Hoofnagle. *Flash Cookies and Privacy*. In: *Intelligent Information Privacy Management, Papers from the 2010 AAAI Spring Symposium, Technical Report SS-10-05, Stanford, California, USA, March 22-24, 2010*. AAAI, 2010. URL: http://www.aaai.org/ocs/index.php/SSS/SSS10/paper/view/1070 (cit. on p. 69).

[Sky19] Skylot. *Jadx - Dex to Java decompiler*. https://github.com/skylot/jadx. 2019 (cit. on p. 79).

[Sla06] Slashdot. *Work Around for New DVD Format Protections*. https://slashdot.org/story/06/07/07/1255224/work-around-for-new-dvd-format-protections. 2006 (cit. on p. 14).

[SMCB12] Benedikt Schmidt, Simon Meier, Cas Cremers, and David A. Basin. *Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties*. In: *CSF*. IEEE Computer Society, 2012, pp. 78–94 (cit. on pp. xii, 7, 95, 100).

[Ste07] Steve Jobs. *Thoughts on Music*. http://web.archive.org/web/20070208225127/http://www.apple.com/hotnews/thoughtsonmusic/. 2007 (cit. on pp. viii, 4, 15).

[Syn] Synamedia. *Synamedia ContentArmor Watermarking*. https://www.synamedia.com/product/contentarmor-watermarking/ (cit. on p. 18).

[Tec18] The Insider Tech Emmy Awards. *70th Award Recipients*. https://theemmys.tv/tech-70th-award-recipients. 2018 (cit. on p. 67).

[The22] The Verge. *Netflix surpasses 200 million subscribers, but has more competition than ever in 2021*. https://www.theverge.com/2021/1/19/22238877/netflix-200-million-subscribers-q4-earnings-bridgerton-emily-paris-cobra-kai-queens-gambit. 2022 (cit. on p. 48).

[Tor23] Tor Project. *Tor Metrics User Stats 2022*. https://metrics.torproject.org/userstats-relay-country.html?start=2022-01-01&end=2023-01-01&country=all&events=off. 2023 (cit. on p. 76).

[tru22] truedread. *netflix-1080p*. https://github.com/truedread/netflix-1080p. 2022 (cit. on p. 63).

[Ver] Verimatrix. *Keep Pirates at Bay with Streamkeeper Watermarking*. https://www.verimatrix.com/products/watermarking/ (cit. on p. 18).

[W3C17] W3C. *W3C Publishes Encrypted Media Extensions (EME) as a W3C Recommendation*. https://www.w3.org/2017/09/pressrelease-eme-recommendation.html.en. 2017 (cit. on p. 67).

[Wat17a] Mark Watson. *Response from Director to formal objection "Turn off EME by default and activate only with express permission from user"*. https://lists.w3.org/Archives/Public/public-html-media/2017Apr/0013.html. 2017 (cit. on pp. 68, 89).

[Wat17b] Mark Watson. *Web Cryptography API*. https://www.w3.org/TR/WebCryptoAPI. 2017 (cit. on p. 89).

[Wid] Widevine. *Widevine DRM*. https://www.widevine.com/solutions/widevine-drm (cit. on pp. 43, 83).

[Wid23] Widevine. *Widevine Integration Platform Shaka Player Tool*. https://integration.widevine.com/player. 2023 (cit. on p. 71).

[Wir16] Wired. *A Bug in Chrome Makes It Easy to Pirate Movies*. https://www.wired.com/2016/06/bug-chrome-makes-easy-pirate-movies/. 2016 (cit. on p. 18).

[WP12] William West and S. Monisha Pulimood. *"Analysis of Privacy and Security in HTML5 Web Storage"*. In: *J. Comput. Sci. Coll.* 27.3 (Jan. 2012), pp. 80–87. ISSN: 1937-4771 (cit. on p. 69).

[WSKV13] Ruoyu Wang, Yan Shoshitaishvili, Christopher Kruegel, and Giovanni Vigna. *Steal This Movie: Automatically Bypassing DRM Protection in Streaming Media Services*. In: *22nd USENIX Security Symposium (USENIX Security 13)*. Washington, D.C.: USENIX Association, Aug. 2013, pp. 687–702. ISBN: 978-1-931971-03-4. URL: https://www.usenix.org/conference/usenixsecurity13/technical-sessions/paper/wang_ruoyu (cit. on p. 55).

[XZL+21] Lei Xue, Hao Zhou, Xiapu Luo, Yajin Zhou, Yang Shi, Guofei Gu, Fengwei Zhang, and Man Ho Au. *Happer: Unpacking Android Apps via a Hardware-Assisted Approach*. In: *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*. IEEE, 2021, pp. 1641–1658. URL: https://doi.org/10.1109/SP40001.2021.00105 (cit. on p. 59).

[zac22] zackmark29. *NetflixESNGenerator*. https://github.com/zackmark29/NetflixESNGenerator. 2022 (cit. on p. 89).

[Zha21] Qi Zhao. *"Wideshears: Investigating and Breaking Widevine on QTEE"*. In: *BlackHat Asia* (2021) (cit. on pp. 18, 55).

COLLEGE MATHS, TELECOMS
DOCTORAL INFORMATIQUE, SIGNAL
BRETAGNE SYSTEMES, ELECTRONIQUE

Université de Rennes

Titre : Briser la confiance : Dissection et Analyse des Impactes Sécurité et Vie Privée du DRM Widevine

Mot clés : DRM, OTT, Widevine, EME

Résumé : Les systèmes opaques présentent un défi unique pour la gestion des droits numériques, ou Digital Rights Management (DRM), Widevine en étant une illustration clé. Malgré son utilisation répandue, ce système DRM a révélé des vulnérabilités importantes dans la protection du contenu propriétaire.

Dans ce manuscrit, nous disséquons le fonctionnement interne de Widevine et analysons comment cette opacité entrave la sécurité tant pour les fournisseurs de contenu, appelés plateforme Over-the-Top (OTT), que pour les utilisateurs. Nous explorons sa mise en utilisation sur les plateformes OTT, identifiant les obstacles qui empêchent son utilisation complète. Les préoccupations en matière de confidentialité sont également abordées, alors que nous enquêtons sur des problèmes d'implémentation au sein des navigateurs internet lors de la standardisation des systèmes DRM par la W3C avec la recommandation Encrypted Media Extension (EME). Ces résultats ne dévoilent pas seulement des problèmes liés à Widevine, mais signalent également des voies de recherche futures dans le paysage DRM plus large, soulignant la nécessité de transparence, de sécurité et de conception centrée autour des utilisateurs.

Title: Bust the Trust: Dissect and Analyze the Security and Privacy Impacts of the Widevine DRM

Keywords: DRM, OTT, Widevine, EME

Abstract: Opaque systems present a unique challenge for Digital Rights Management (DRM), with Widevine DRM as a key illustration. Despite its widespread usage, this DRM system has revealed significant vulnerabilities in protecting media content.

In this manuscript, we dissect Widevine inner workings and analyze how this opacity hampers security for both companies and users. We explore its usage across OTT platforms, identifying barriers that prevent its full utilization. Privacy concerns are also addressed, as we investigate real problems due to browser misimplementation within DRM systems standardization by the W3C with Encrypted Media Extension (EME) recommendations. These findings not only expose Widevine challenges but also signal future research avenues in the broader DRM landscape, emphasizing the need for transparency, security, and user-centric design.