Hardware Support for DRM: Can Trusted Execution Environments save us from System Lock-down?

Oliver Schmidt
TU Dresden
oliver.schmidt3[at]mailbox.tu-dresden.de

***Abstract*—Nowadays, all major publishers of popular media content mandate the usage of DRM copy protection for their published works. But as preventing digital content from being copied is a difficult problem, the current approaches for copy protection involve locking down systems, criticized as endangering general computation and user freedom.**
This survey looks at the available hardware support technologies for providing hardware trust anchors or trusted execution environments, ranging from Secure Boot and TPM to Intel SGX and TrustZone, and evaluates their usage in the DRM architectures of Android and the HTML5 EME browser stack. Showing shortcomings in authenticity checking of components, different levels of required lock-down, and output protection of all covered architectures, we find that the usage of trusted execution environments reduces the required system lock-down for DRM protection. But many other negative effects of DRM can not be solved with this approach.

***Index Terms*—Security, DRM, Trusted Execution Environments, SGX, TrustZone, TPM, HTML5 EME**

I. Introduction

Since early on in the so-called "information economy", publishers have tried to limit the distribution of their digital goods to the buyers only. In his keynote "The coming war on general computation" at the 28th Chaos Communication Congress [1], Cory Doctorow outlines the development of copy-protection mechanisms from first including uncopiable bad sectors to the floppy disks on which programs were distributed or tying the execution to dongles and license keys, to encrypted music and video files protected with dedicated Digital Rights Management (DRM) schemes.
But these attempts were mostly based on obscuring the protection mechanism used, thus being vulnerable to circumvention through reverse-engineering, patching out the protection mechanism or retaining the supposedly-secret media decryption keys even after the usage license expired. The renowned IT security columnist and expert Bruce Schneier commented on these attempts: "Digital files can not be made uncopyable, any more than water can be made not wet." [2] The underlying vulnerability of all these copy protection schemes was that they were attempting to "[…] figure out how to stop computers from running certain programs and inspecting certain files and processes." [1] But as modern day computers are mostly general computation machines, they are inherently based on copying data and computing on them.

Doctorow warns that all countermeasures trying to ensure copy protection of digital content are going to result in creating "appliances", which are still general purpose computers, but locked down with "some combination of rootkits, spyware, and code-signing to prevent the user from knowing which processes are running, from installing her own software, and from terminating processes that she doesn't want." [1]
Additionally, international agreements based on the World Intellectual Property Organization (WIPO) Copyright Treaty [3], its most prominent implementation being the US-American Digital Millenium Copyright Act (DMCA), make the circumvention of "technical protection measures" such as DRM illegal.

So right now we are in a situation, where all major publishers of e.g. video[1] and audio[2] content, and video games[3] require the usage of DRM for protecting their published works. Thus right now, all software using this content has to be proprietary and whole platforms are being locked down more and more. This development is most apparent in non-PC platforms like mobile devices, where unlocking the bootloader (if even possible) results in deletion of DRM keys of the device. [5]

But in recent years modern CPU architectures have introduced special hardware-backed Trusted Execution Environments (TEEs) to provide a secured environment for security-critical code to be executed in isolation from the main *untrusted* operating system (OS). Can these TEEss and other special hardware trust-anchors provide the possibility to present DRM-secured content in an otherwise open and open source system? Or does DRM only work on completely locked-down systems?

In II we first give an overview of common technologies used for providing trust anchors for running systems or isolating software into a trusted environment. Afterwards, III covers existing DRM architectures already utilizing TEEs.

[1] all Hollywood film publishers require adhering to certain protection standards like [4]

[2] although legal music file purchases are mostly DRM-free, the consumption model of streaming has brought back DRM to platforms like Spotify or Deezer

[3] Valve's Steam platform integrates its own DRM into sold games; other publishers and gaming consoles have their own DRM systems as well

Aiming for the usage on systems as open as possible, IV looks at the security of the presented architectures on such systems and at last takes a look at other problematic effects of DRM usage.

II. Background: Trust Anchors and Trusted Execution

This section first gives an overview about technologies used for having a trust anchor in a running system. Although these approaches are often used for locking down whole systems, they may also provide the basis for building more open systems by tying trust to hardware anchors instead of a locked-down software stack.
Afterwards we cover the technologies dedicated to providing a TEE in modern processor architectures.

A. UEFI Secure Boot

Secure Boot is a functionality of the Unified Extensible Firmware Interface (UEFI) boot firmware component [6] to allow only the launch of authenticated boot images. To achieve that, boot images can be signed with X.509 certificates. Only if the image verifies correctly against a key stored in non-volatile firmware memory or against an entry in an explicit allow list of signatures, it is launched by the firmware. This first check on which bootloader or OS image to launch can be the anchor of a trust chain, if each consecutive execution step also checks the authenticity of software to be launched. The allow and deny lists can be updated from the running OS and deploying own custom platform keys for verification can be possible through setup-mode of UEFI.

Firmwares adhering to the UEFI standard are the dominant system software for the PC platform (including x86 devices like servers, laptops and mobile devices, but also more and more devices with ARM chipsets).

B. TPM

Trusted Platform Modules (TPMs) are dedicated hardware components offering a variety of security services to a system [7]:
TPMs provide **secure storage** inside the module to store sensitive data like cryptographic keys in such a way that it can not be leaked to or stolen from processes running on the main CPU.
These stored keys can be used to execute cryptographic operations on behalf of the system, without the keys getting transferred to the outside of the module.
The **attestation** functionality can attest the identity of software by calculating a hash sum of the supplied binary, signing it with its internal key to prove a certain system configuration or software version to third parties having a corresponding verification key. When pre-deployed with a unique key, all previously mentioned functionality can be combined to provide a **unique machine identity** and form a hardware trust anchor for running systems.
A **continuous trust chain** can be built from the boot-up on if all software components, starting from the firmware on, let the TPM attest the processes to be launched. Based on reference signatures stored in the module's secure storage, each component can then decide to allow or refuse the next software to launch. A similar boot policy this attestation-checking during boot is **authenticated booting**. Again the TPM calculates the checksum of each boot stage but does not enforce any signature checks and only stores the results inside its secure internal registers, from where the system status can be queried later.[8] In contrast to UEFI Secure Boot (II-A), it is also possible to securely attest and launch code using special *late-launch* CPU instructions introduced into AMD and Intel chipsets without providing a continuous trust chain from the firmware on [8].
Additionally, TPMs provide important support functionality for cryptographic algorithms like **secure counters**, a **secure clock** for peripherals and a **secure source of entropy**. [7]

C. Trusted Execution Environments

To allow separating the execution of security critical code from the large untrusted codebase of the rest of the system, all major processor vendors have introduced so called TEEs. These are separated co-processors, either dedicated or virtualised, providing a computing environment with at least its own computing resources, registers and memory (areas) isolated from the untrusted parts of the running system.

This section is mostly not based on the vendor's refrence manuals but on other papers reviewing the applicability of such environments. [8], [7], [9]

1) ARM TrustZone: ARM trust zone [10] is a set of security extensions for processors, currently available for the ARMv8 architecture. Additionally, the processor vendor AMD has announced to include TrustZone co-processors into some of their x86 processors. It divides the physical processing environment into two separate *worlds* that can be switched. The **secure world**, being the initial processor state after reset, and the **normal world**, where all legacy and non-critical code is run. Each world provides its own runtime environment with (virtually) dedicated registers, memory, processor, caches and interrupts. Additionally, each world runs its own operating system, with general purpose OSs running in the normal world, while the secure world tends to use special purpose OSs like the *Trusty OS* [11] used by the Android Open Source Project.

How these dedicated virtual resources are actually backed by physical ones depends on the implementation of the system on a chip (SoC). Resources can either be strongly partitioned (e. g. memory), shared between worlds (often the case for caches, processor) or assigned exclusively to one of the worlds (most I/O devices, separate register banks).
The memory is strongly partitioned on boot-up, rendering the worlds inaccessible to each other. This memory

isolation is supported by marking the secure world with an additional bit at the hardware level.
Communication between the worlds is made possible by the *secure monitor* processor mode, controlled by a dedicated register. This monitor mode can implement context switches between the worlds as it can access copies of non-secure registers from the secure context. ARM CPUs always boot first into the secure world and hand over control to the non-secure world after initialization. Later on, the normal world can use a special *secure monitor call* instruction to give control to the secure monitor mode, performing a context switch. Interrupts can directly map into the secure monitor mode.

With the memory only being separated but not encrypted, TrustZone's security relies on having a connection to its memory which can not be eavesdropped upon.

2) Apple SecureEnclave Processor: Although Apple's own SoCs use the ARM architecture, the company has decided to use their own secure co-processor solution instead of TrustZone (II-C1). Their *Secure Enclave Processor* is a dedicated processor, running an L4 microkernel, communicating with the main CPU over a bus system and accessing the system's memory using inline-encryption. [8]

3) Intel SGX: Intel's Software Guard Extensions (SGX) are a method to launch multiple trusted components into their own fully isolated *enclaves* and thus, according to [8], can be seen as a more advanced version of the late-launch approach (II-B). Enclaves can be scheduled by the OS like normal processes, but code and memory of enclaves are only visible from the inside. Enclave memory resides in the common system's DRAM but is transparently encrypted. Additionally, SGX includes attestation of code running within an enclave similar to TPM, but does not provide other TPM features like secure non-volatile storage.
As only userland code running in ring 3 can be run inside enclaves, relying on the OS for scheduling and resource management, SGX is vulnerable to side-channel attacks [12].

III. Hardware Support for DRM systems

So how can these hardware security mechanisms be used to support DRM mechanisms and make them run securely within a still open platform?

A. Android DRM architecture

The Android platform provides its own DRM framework[13], providing an API for applications to communicate with a *DRM manager* over inter-process communication (IPC). This DRM manager then runs plug-ins managing the actual DRM schemes as separate processes for isolation purposes.
The level of protection provided by the DRM plug-in varies depending on the plug-in itself and on the capabilities of the hardware platform. Plug-ins may rely on secure boot for a verified chain of trust from the firmware level on, use protected output mechanisms provided by the hardware platform and even run the programs inside a TEE.

Plug-ins are automatically loaded when they are placed into the `/system/lib/drm/plug-ins/native/` directory. One issue we see here is that there is no mention of authenticity checking of plug-ins in the documentation. [13] This can enable DRM plug-ins to claim to be able to decrypt a certain stream and thus at least result in the user not being able to decrypt their media with the proper add-on. Additionally it might also open up attack vectors for the communication with a license server, as shown later.

One cross-platform DRM plugin solution is **WiDevine** [14], currently owned by Google. It provides native solutions for the Android, iOS and HTML5 platform. Thus the DRM decryption process is very similar to the one specified in HTML5 EME and will be covered in section III-B in more detail.
The reason we look at the example of the WiDevine Android plugin is that it supports different security levels, depending on the use of hardware security mechanisms [14]: For security level 1, "[a]ll content processing, cryptography, and control is performed within the Trusted Execution Environment (TEE)." This includes that even the decrypted video frames are to be passed to the graphics hardware using a secured mechanism.
This is not the case for security level 2: There only the cryptographic operations are done within a TEE, but the decrypted video content is passed to processes running outside of a secure environment for further decoding, demuxing and post processing. But still all cryptographic keys remain secured within the TEE.
And security level 3 is provided on devices without a TEE at all, where "[a]ppropriate measures may be taken to protect the cryptographic information and decrypted content on host operating system", which can be considered as a system-level lock down.

The architecture overview does not mention how to persistently store the acquired decryption keys. At the same time we know that apps[4] are offering functionality like an offline mode, requiring persistent storage of keys.

B. HTML5 EME

To be able to provide DRM support for media content in the web, the Encrypted Media Extensions (EME) have been standardized by the World Wide Web Consortium (W3C) [15]. EME are not a DRM system themselves, but provide a standardized JavaScript API for mediation between HTML5 media elements and so-called Content Decryption Modules (CDMs). These CDMs are not standardized in EME but are proprietary modules managing the actual DRM scheme including license management, key retrieval, decryption and (optionally) decoding.

[4]example: Netflix for Android https://help.netflix.com/en/node/54816

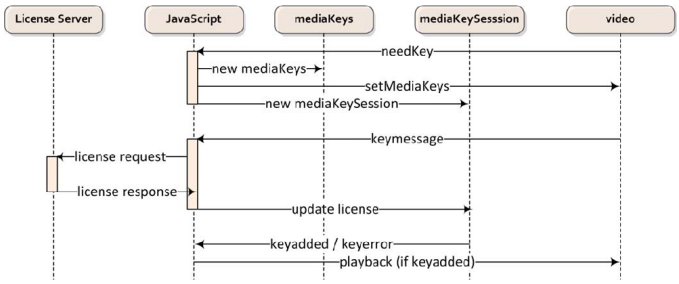

Fig. 1. Sequence of information exchange between EME components, source [9]

In [9] Livshits et al. analyse current web browsers and their EME implementations for potential vulnerabilities and propose an architecture to back CDMs with TEE-usage – not only for higher security but also to fulfil the requirements of media publishers of having hardware-backed copy protection.

EME flow: For the playback of DRM protected media, information has to be exchanged between the web application and its media file, the CDM and license/ key servers.

First the browser's media stack parses the embedded media file and discovers the *key id* embedded into the media file's metadata. This fires a *needkeys* event including some initialization data to the web application which then creates the *mediaKeys* and *mediaKeySession* for a specific key system. After the initialization data is pushed to a CDM implementing the key system, it generates a *keymessage* for a license server. The message is then sent to the license server by the browser, passing the response back to the CDM as well which decrypts the received license and updates the *mediaKeySession*.
After a *keyadded/keyerror* event fired back to the web application, indicating the success status of the license retrieval, it can finally initiate the playback of the media file, causing the CDM to decrypt it using the received license key.

Hereby all javascript events fired from the CDM to the web application contain byte buffers, which the web application running in browser context passes around and sends them to the respective servers. But these byte buffers are usually encrypted by the CDM and thus incomprehensible to the browser handling them. [16]

One interesting side-note is that due to the usage of the *ISO Common Encryption* standard for encryption of the media files, the same file can be decrypted by different CDMs implementing different DRM schemes. As the license format and content of the encrypted byte buffers are proprietary, differing between various DRM schemes, this requires the content provider to operate the respective license servers for each scheme. But having acquired a license, each CDM can then decrypt the media file accordingly.

Integration with a TEE: The EME standard only mandates the content decryption to take place in the CDM, so the media file might as well be passed back to the browser's media stack for decoding and rendering. Livshits et al. identified this as potential vulnerabilities of a DRM scheme, as the decrypted media data might be easily grabbed from the browser context by abusing security vulnerabilities, adding add-ons (browser extensions) to the running browser or grabbing the content from pipes between the media stack's components. [9] As we are looking at the usage of DRM in an as-open-as-possible system, we would like to add the possibility of just changing the source code of an available open source browser[5] to additionally capture all media data passed back to it for decryption.
As a countermeasure, they propose to move both the CDM as well as the whole media decoding and rendering stack to a TEE, making the browser player part of the trusted computing base (TCB). For that purpose they extend the EME architecture by adding a CDM proxy component running in browser context, which has to forward EME calls to the real CDM running within the TEE. They specify how to run the CDM using Intel SGX (see II-C3 or using ARM TrustZone (see II-C1).
On systems with SGX, the CDM is launched inside a secure enclave created by the browser. When utilizing TrustZone, inside the normal world the browser notifies its OS that the license- or metadata needs to be exchanged to the secure world. Initiating a *secure monitor call*, the data is transferred to the secure world where the CDM can now create a license request or can decrypt the media content. To prevent the leakage of the decrypted media during rendering and display, the media pipeline has to be secured as well. In [9] this is done for the graphics pipeline by utilizing measures available on the respective hardware platforms: On *SGX platforms*, Protected Audio Video Path (PAVP) is used to create a video surface within the browser context. From the SGX context DXVA 2.0[6] is used to exchange a key with the GPU with which the DRM-decrypted media content is re-encrypted before being sent to the surface. Inaccessible to the browser context, the media is decrypted and decoded directly on the GPU and then sent to the display device.
The audio pipeline is not mentioned, but with combined transmission of audio and video signals like in HDMI or DisplayPort we consider this covered as well.
On platforms with *ARM TrustZone*, memory sections can be dedicated to the secure world only and access to these sections can be limited to certain devices. This ability is used to store raw video frames, decoded in software running inside the TEE, to be accessed by the GPU only. Additionally, a session between the CDM inside the TEE and the GPU can be authenticated with certificates, enabling video decoding directly on the GPU similarly to

[5]both Chromium and Firefox are open source browsers supporting the EME standard

[6]*DirectX Video Acceleration*, a proprietary API available on Microsoft Windows and Xbox

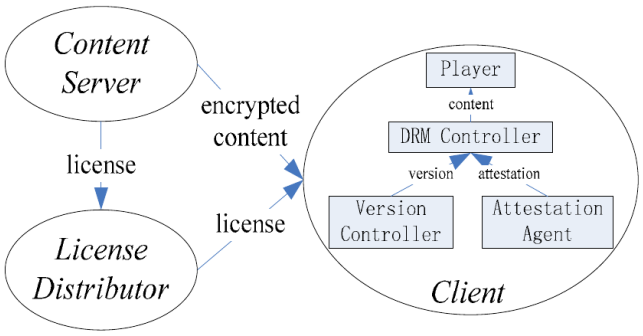

Fig. 2. high-level architecture of TBDRM components, source: [17]

the approach taken on SGX systems.

C. TBDRM

In 2009 Yu et al. have proposed a DRM architecture based on TPM, called *TBDRM*. [17] Their approach relies on multiple components checking each others authenticity, relying on the attestation functionality of TPM for constructing a hardware trust chain and authenticity checking, and on TPM key management functionality for binding content to a certain player component.

The high-level architecture of TBDRM is outlined in Fig. 2. The encrypted content itself can be delivered independently from the license, which consists out of a usage policy for the content, its decryption key and some metadata for describing and identifying the matching content.

The trust basis for all other client components is provided by the *Attestation Agent (AA)*. This component can attest the authenticity and identity of the *Version Controller (VC)*, which ensures and attests freshness of a license, the *DRM Controller (DC)* handling the actual content decryption and policy enforcement, and the actual media *player* component.

When requesting a license, the License Distributor (LD) decides whether to give a license based on the attested identity of the DC and the license version requested. At playback, the trustworthiness of the player component is first attested to the DRM Controller by the Attestation Agent, the same is done for the Version Controller afterwards. If all components are trustworthy to the DC and the VC also has verified the freshness of the license, the DC checks the requested usage permission against the usage policy of the licenses. If access can be granted, the media content is decrypted with the symmetric key provided by the license and passed to the player. If the usage policy mandates a version bump of the license (e. g. for enforcing a playback number limit), this is done by the VC after playback.

For implementing this architecture, the authors proposed using several TPM functionality for ensuring the security (persistent control of content, license integrity/ confidentiality/ freshness) of the DRM system.

In their prototype they ensure a **trust chain** from the boot on to each DRM scheme component by measuring each system boot step, starting at the hardware trust anchor provided by the TPM. Measuring means the TPM computing the hash of a component and storing the result into its internal platform configuration registers, from where it can be retrieved later. First the TPM measures the hardware platform configuration, the bootloader containing a *kernel measuring agent* and the kernel containing an *application measurement agent* serving as the AA component. A full trust chain can then be established by checking all component's measurements from the respective TPM registers. Additionally, the kernel contains the VC component and the part of the DC responsible for verifying the trustworthiness of all other components, turning all components of the system stack from the hardware to the kernel itself into parts of the trusted computing base. The player itself and the DC part responsible for interpreting and deciding on policy rules are running in the user-space and are isolated from each other. The application measurement agent and the user-space **isolation** are done based on a Linux Security Module running in the kernel, the latter by preventing processes from accessing other processes' address space e. g. using ptrace.

The **bind mechanism** for making license data only accessible to certain components is based on the TPM key management mechanism *immigratable key* and *key certification*. For binding a license to the current DC, a new immigratable asymmetric keypair is created. When acquiring a license, the public key of that pair is certified by the TPM and sent to the License Distributor. After verification, the LD uses hybrid encryption for the license data by encrypting it symmetrically with a fresh key and encrypting that key asymmetrically with the received public key of the client. Thus the sent-back license can only be decrypted and used by the trusted DC. The TBDRM proposal does not specify whether licenses can be saved to persistent storage.

Freshness of licenses is ensured within the VC by using monotonic counters of the TPM. The state of each counter is bound to the VC and can be stored on disk for persisting. While the paper itself does not go into detail which cryptographic algorithms are used for persistence, we want to emphasize the usage of authenticated encryption for storing such counter values. Malleable or unauthenticated encryption like plain RSA would not be sufficient for storing such a small, easy guessable counter value.

The TBDRM paper builds on hardware TPMs conforming to the TPM 1.2 specification. This version of TPM has been deprecated for several years now due to its cryptographic algorithms having become less secure and its different implementations being ambiguous. The successing version of the specification, TPM 2.0, is backwards incompatible but provides roughly the same functionality. Thus porting the TBDRM approach to work with TPM 2.0 should be doable.

A TPM 2.0 based variant of TBDRM can also become interesting for vendors not wanting to include a dedicated TPM chip into their device, but having a CPU with TEE functionality: With **fTPM** [7] Microsoft researchers have presented an implementation of the TPM 2.0 specification using Trusted Execution Environments, specifically ARM TrustZone (described in II-C1) as their basis, complemented with some additional hardware components. They first analyse the shortcomings of TEEs compared with the functionality provided by TPMs. The shortcomings of the ARM TrustZone TEE relevant for TBDRM are its lack of secure storage (required for key storage), lack of secure persistent counters (used by the VC) and lack of secure entropy source (used for key generation). For overcoming these shortcomings, they took different approaches:

- *adding additional hardware*: replay-protected, authenticated storage (eMMC with replay-protected memory block), hardware fuses available only to the secure world as seeds for key generation, a physical entropy generator
- *design compromises*: no long-running TEE processes to increase system stability – cooperative checkpointing for splitting up RSA key generation into multiple short steps
- *modified TPM 2.0 semantics*: changing semantics of services slightly while still maintaining verifiable security guarantees

As fTPM requires additional hardware components to be included already during system design, it can not be applied to existing devices lacking these components. But if these small additional requirements are taken care of during system design, hardware-complemented TEE features of modern CPUs can provide TPM-compliant services. This is the case on all ARM-based devices by Microsoft shipped with the Windows OS, where the required TPM functionality is provided by fTPM.

IV. Shortcomings

A. shortcomings on open systems

As the aim of this paper is to evaluate the feasibility of providing DRM protection on otherwise open platforms, in this section we look at what security challenges arise for the presented DRM architectures, which parts of the system need to be secured and which of these can remain open.

1) authentication at license server: Both the WiDevine architecture [14] and the TEE-backed HTML5 EME approach leave an important attack vector uncovered: It is not specified how CDMs authenticate against the license server they get the decryption key from. While all information between the CDM and the license server is encrypted and thus meaningless to the browser or other software in the middle of the communication, on an open system it should be possible to read, reverse-engineer and even manipulate the CDM binary[7], as they have to be stored somewhere (SGX and TrustZone do not provide secure permanent storage) or transmitted to the computer for installation. Livshitz et al. even mention reverse-engineering and manipulation in the security objectives of their paper [9], but do not provide any countermeasures.
Although encrypting the communication between CDM and license server without parties in the middle being able to eavesdrop on the license key transferred is indeed possible, e.g. using a Diffie-Hellman key exchange for generating a session key[8], the license server can not check the authenticity of the CDM it is communicating with. Including a private authentication key into the CDM binary is insecure as the key could be recovered from attackers by reverse engineering. And if there is no need to authenticate the CDM against the license server or modified CDMs are still able to do that using an extracted private key, a modified CDM can either deliberately leak the key or the decrypted content to the untrusted OS.

As a countermeasure we suggest the usage of attestation mechanisms e. g. remote attestation of SGX or (f)TPM for verifying the integrity of the used CDM. Especially for SGX though this is only a starting point and special care needs to be taken for remote attestation to be successful, as pointed out in [18].

The Android DRM framework seems to put emphasis on locking down the whole system with a trusted boot chain instead, as stated in the Android documentation: "The combination of hardware security functions, a trusted boot mechanism, and an isolated secure OS for handling security functions is critical to providing a secure device." [13]
This is also apparent in the light of several apps using the Android DRM framework not being available on rooted devices. One example is the Netflix app, which uses the WiDevine DRM scheme. [19] This app can not be installed from the Play Store[9] on rooted Android devices, though already installed apps or ones installed through side-loading still continue to function. This is the case because installation from Play Store for this app is tied to the device's *SafetyNet* status, which tries to detect tampering with the device's software. The Android DRM framework itself though also offers APIs for getting the security status of a DRM plug-in (see III-A), including whether a DRM plug-in fully uses the TEE of a device. Due to the ambiguous restriction enforcement it remains unclear whether banning the store installation of such apps on rooted devices is a sign of Google not fully trusting the security of TEE-backed DRM even on open devices or if

[7] under the WIPO copyright treaty or the DMCA this is illegal, but still technically possible

[8] although this could become a bit cumbersome in HTML5 EME as the browser has to relay all steps of the key exchange before actual payload information can be sent

[9] Google's app store for the Android platform, by far the most popular one and default on Google-certified devices

this is just a side effect of administrational policies of the store. But at least this is not a sign pointing towards the acceptance of DRM on open platforms.

2) size of the trusted computing base: The advantage of the approaches where Content Decryption Modules run inside a TEE is that, given the TEE technology itself is secure, authentication issues like the ones just described can be overcome and the cryptographic primitives used are semantically secure, only the TEE running the (proprietary) CDM itself is part of the TCB. All other software on the untrusted (virtual) processor, even the operating system, might be modified freely as all sensitive data (plain content, cryptographic keys) are never accessible outside of the secured environment. When using ARM TrustZone though, care needs to be taken of other code running within the secure world. As there is only one secure world available, all programs running within the TEE have to share the same environment, on top of a special purpose operating system like the Trusty OS [11] used on Android devices. There, processes running on top of the OS are isolated from each other via address space separation, utilizing the Memory Management Unit (MMU). So at least the CDM and the trusted OS it is running on are part of the TCB. In the case of Trusty, even all processes running on top are bundled with the OS, signed, and then verified by the firmware at boot similarly to Secure Boot. And still all vulnerabilities in code belonging to the TCB are potentially security critical if exploitable, one example of it shown with a vulnerability in the TEE OS itself. [20]

For the TBDRM approach though the size of the TCB is even larger, including the whole kernel and TBDRM's userland components. Firmware and bootloader are part of the trust chain and must not be modified for the DRM system to work, but are not a crucial part of the TCB as the kernel running on top is measured by the TPM independently. As the TPM provides important the cryptography and trust services to the system, it is part of the TCB as well.
As measured boot is used, modified software components are not prevented from running but the DRM functionality would not work on systems not identified as trustworthy. Thus a dual-boot system can be imagined, where in one of the systems the kernel can be freely modified while loosing the ability to play DRM secured content, while the unchanged other system retains the ability of DRM-protected playback.

When using UEFI SecureBoot, there needs to be a trusted path from the firmware to the trusted DRM software (e. g. a CDM) to be able to verify the authenticity the component. Additionally, appropriate measures need to be taken to ensure that data inside the trusted DRM component can not be read by untrusted processes. This results in a TCB comprising the whole operating system and all higher-privilege user land components. Effectively, this results in a lock-down of the whole system.

In practice, Microsoft requires devices [21] shipped with the Windows OS to have Secure Boot enabled by default, verifying boot images against Microsoft's key. This requirement has been criticized as it might lead to lock-down of consumer hardware and the inability to install alternative operating systems on it.
Earlier versions of the requirements mandated an option to disable Secure Boot on x86 devices while also mandating that it must always be enabled on ARM-based devices. [22] At least on x86 devices as there is a shim bootloader signed by Microsoft [23], enabling the launch of other unsigned boot images. The signature of this specific binary could though be put on the firmware's deny list through updates.

3) media output on peripherals: The secured rendering pipeline used in III-B only reaches to the GPU. But how to transfer the decoded audio and video data to the actual display device and speakers?
For directly attached screens on devices like smartphones, tablets or laptops relying on the display bus being physically hard to eavesdrop on might be a valid compromise. But as soon as it comes to attaching external display devices over standardized connectors (HDMI, DVI, DisplayPort), direct output grabbing becomes a threat. With these video output standards transmitting digital data, the quality loss during grabbing is neglectable. The state-of-the-art mechanism for DRM protection during transmission is High-bandwidth Digital Content Protection (HDCP), which encrypts the content on its way to the display device. But with the master key having leaked and decryption hardware being available [24], HDCP can be considered as broken.

B. Further Downsides of DRM

While the usage of TEEs can indeed increase the security of DRM schemes while at the same time taking away the need for total system lock-down, there are quite a number of other issues with the usage of DRM protection for media. Most of them are less purely technical but concern the social and political effects of its technical properties.

The designed purpose of DRM systems is to restrict the usage of protected media files to follow the exact policies of the publisher – and that is exactly what it does. The problem here: The outlined policies of the publisher do not always cover legal or legitimate use cases of the media. In their campaign leaflet on DRM and its downsides [25], the Free Software Foundation Europe outlines several problems of DRM usage:
DRM protection of media content undermines several copyright exemption and customer rights. While most jurisdictions know exemptions for lending works out to friends and family or making private copies for them, this copyright exception is not granted when so-called *effective measures* of copy protection are taken by the publisher. All DRM mechanisms are considered as such and are illegal to circumvent, taking away these user rights. At the same time we notice that, at least in Germany, collector societies

are still charging and collecting remuneration fees for this exception customers are not allowed to use any more. A similar copyright exemption is the liberty of quotation. But with DRM protection, it is not possible to e. g. cut a certain video sequence from a film to embed it into your own work.
In the United States, there is even the concept of *fair use* allowing the remix of existing works to create new works of art. As DRM systems prevent editing and further processing of the content, this right is also prevented from being executed.
In DRM systems only certain pieces of software are allowed to process the protected content – in the case of TBDRM these are the players trusted by the license distributor. Often publishers forget the reliance of disabled people on certain assistance software. Such screen readers, image or audio enhancers are rarely on the list of trusted software able to process the protected content and thus can not enable disabled people access to the content.

DRM protection is also a threat to the availability of content, both in the short and in the long term. Recently, an anecdote [26] raised awareness for this again: After moving from Australia to Canada, a person lost their ability re-download several purchased films. This happened due to regionally-limited licensing deals of the publisher. While already downloaded film files could still be played back, because of DRM protection they were tied to a single device only. Playback of the legally bought films on other devices would have required a re-download, which was not possible in that jurisdiction.
This illustrates the downsides of DRM protected content always being bound to something out of the control of the consumer who bought it: For acquiring the required decryption keys, license servers need to be online, reachable and supporting the technology required for the media to be played back. Additionally, these proprietary DRM schemes are always bound to a certain technology platform (e. g. ARM TrustZone and Android) with only their vendor having the ability to make content bought for a previous technology available on a new platform. While the *Common Encryption* standard is a step into the right direction, we are still in a situation where content regularly becomes unavailable because of companies going out of business, deciding not to make old content available on new platform or switching off licensing servers[10]. This is also a threat to preserving cultural heritage, as even personal backups of the media files are useless without the non-exportable respective keys.

The seemingly endless possibilities of restrictions motivated companies to also offer business options requiring heavy user tracking. For example, *node locking* – binding content to a specific computer only – causes several DRM schemes to create detailed profiles of the computer it is running on and the user's activities. Because of that Mozilla decided in their implementation of the HTML5 EME standard to run CDMs only sandboxed, limiting tracking and privacy intrusion. [28]
As CDMs are proprietary[11], both the amount of user tracking they do and their security can not be evaluated, as under the WIPO agreement reverse-engineering copy-protection mechanisms is heavily restricted. With the availability of Trusted Execution Environments and the usage of attestation, not requiring CDM binaries to contain secret data any more, it should be possible to create open source CDMs which can be built reproducibly.

[10]example: in 2008 Microsoft switched off their "Plays for Sure" licensing servers [27]

V. Summary

As presented on the examples of the covered DRM schemes, the usage of hardware support mechanisms – especially of Trusted Execution Environments – can enable reliable content protection on systems without having to lock them down completely. All presented approaches had shortcomings in the areas of authentication and authenticity checking, size of the locked-down system base (TCB) or actual output mechanisms, but when combining ideas from these different schemes the result is more resilient to threats.

However, the usage of DRM implies several other problems that are not solved by the usage of hardware support.

VI. Glossary

AA	Attestation Agent
API	Application Programming Interface
CDM	Content Decryption Module
CS	Content Server
DC	DRM Controller
DMCA	Digital Millenium Copyright Act
DRM	Digital Rights Management
EME	Encrypted Media Extensions
HDCP	High-bandwidth Digital Content Protection
IPC	inter-process communication
LD	License Distributor
MMU	Memory Management Unit
OS	operating system
PAVP	Protected Audio Video Path
SGX	Software Guard Extensions
SoC	system on a chip
TCB	trusted computing base
TEE	Trusted Execution Environment
TPM	Trusted Platform Module
UEFI	Unified Extensible Firmware Interface
VC	Version Controller
W3C	World Wide Web Consortium
WIPO	World Intellectual Property Organization

[11]The WiDevine architecture overview even mentions this as a key security concept: "This unique mix of open source and protected source enables WiDevine DRM to make it easy to create custom playback applications that are encrypted and secure." [14]

References

[1] Cory Doctorow, "The coming war on general computation," 28. Chaos Communication Congress, Berlin, Dec. 2011, transcript: https://joshuawise.com/28c3-transcript. [Online]. Available: https://media.ccc.de/v/28c3-4848-en-the_coming_war_on_general_computation

[2] Bruce Schneier, "Crypto-Gram: May 15, 2001 - Schneier on Security," May 2001. [Online]. Available: https://www.schneier.com/crypto-gram/archives/2001/0515.html#3

[3] World International Copyright Organization, "WIPO Copyright Treaty," p. 9, Dec. 1996.

[4] MovieLabs Inc., "MovieLabs Specification for Enhanced Content Protection, v1.2," Aug. 2018. [Online]. Available: http://www.movielabs.com/ngvideo/MovieLabs_ECP_Spec_v1.2.pdf

[5] Sony Developer World, "Unlock Bootloader - Open Devices - Sony Developer World." [Online]. Available: https://developer.sony.com/develop/open-devices/get-started/unlock-bootloader/

[6] Unified EFI Forum, Inc., "UEFI Specification version 2.7 errata A," p. 2575, 2017.

[7] H. Raj, S. Saroiu, A. Wolman, R. Aigner, J. Cox, P. England, C. Fenner, K. Kinshumann, J. Loeser, D. Mattoon, M. Nystrom, D. Robinson, R. Spiger, S. Thom, and D. Wooten, "fTPM: A Software-Only Implementation of a TPM Chip," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, 2016, pp. 841–856. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/raj

[8] H. Hartig, M. Roitzsch, C. Weinhold, and A. Lackorzynski, "Lateral Thinking for Trustworthy Apps." IEEE, Jun. 2017, pp. 1890–1899. [Online]. Available: http://ieeexplore.ieee.org/document/7980129/

[9] D. Livshits, A. Mikityuk, S. Pham, and A. Shabtai, "Towards Security of Native DRM Execution in HTML5," in *2015 IEEE International Symposium on Multimedia (ISM)*, Dec. 2015, pp. 411–416.

[10] ARM Limited, "ARM Security Technology Building a Secure System using TrustZone Technology," p. 108, 2005.

[11] "Trusty TEE." [Online]. Available: https://source.android.com/security/trusty/

[12] PEINADO, M., XU, Y., and CUI, W., "Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems." in *Proc. of the 36th IEEE Symposium on Security and Pri- Vacy (Oakland)*, 2015.

[13] "Android DRM Framework." [Online]. Available: https://source.android.com/devices/drm

[14] Google, "Widevine DRM Architecture Overview v1.2," Mar. 2017. [Online]. Available: https://storage.googleapis.com/wvdocs/Widevine_DRM_Architecture_Overview.pdf

[15] W3C, "Encrypted Media Extensions - www.w3.org/," Sep. 2017. [Online]. Available: https://www.w3.org/TR/encrypted-media/

[16] "What is EME? - hsivonen.fi/." [Online]. Available: https://hsivonen.fi/eme/

[17] A. Yu, D. Feng, and R. Liu, "TBDRM: A TPM-Based Secure DRM Architecture." IEEE, 2009, pp. 671–677. [Online]. Available: http://ieeexplore.ieee.org/document/5283799/

[18] Swami Yogesh, "Intel SGX Remote Attestation is not sufficient," Las Vegas, 2017. [Online]. Available: https://www.blackhat.com/docs/us-17/thursday/us-17-Swami-SGX-Remote-Attestation-Is-Not-Sufficient-wp.pdf

[19] Corvin Davenport, "Netflix confirms it is blocking rooted/unlocked devices, app itself is still working (for now)," May 2017. [Online]. Available: https://www.androidpolice.com/2017/05/13/netflix-confirms-blocking-rootedunlocked-devices-app-still-working-now/

[20] D. Shen, "Exploiting Trustzone on Android," 2015, p. 7.

[21] "Microsoft Hardware Certification Policies and Processes," 2014. [Online]. Available: download.microsoft.com/download/4/D/D/4DD894CD-62C8-488F-944D-4E5F8BA40114/hardware-certification-policies-processes-hck2-1.docx

[22] G. Moody, "Is Microsoft Blocking Linux Booting on ARM Hardware?" [Online]. Available: https://www.computerworlduk.com/it-business/is-microsoft-blocking-linux-booting-on-arm-hardware-3569162/

[23] Matthew Garrett, "Announcing the Shim review process." [Online]. Available: https://mjg59.dreamwidth.org/47438.html

[24] B. Lomb and T. Güneysu, "Decrypting HDCP-protected Video Streams Using Reconfigurable Hardware," in *2011 International Conference on Reconfigurable Computing and FPGAs, ReConFig 2011, Cancun, Mexico, November 30 - December 2, 2011*, P. M. Athanas, J. Becker, and R. Cumplido, Eds. IEEE Computer Society, 2011, pp. 249–254. [Online]. Available: https://doi.org/10.1109/ReConFig.2011.24

[25] Free Software Foundation Europe and Joe McNamee, "DRM: The Strange, Broken World of Digital Rights Management," 2012. [Online]. Available: https://edri.org/wp-content/uploads/2013/10/paper04_web_20120205.pdf

[26] J. Bergmayer, "It's Always DRM's Fault," Sep. 2018. [Online]. Available: https://www.publicknowledge.org/news-blog/blogs/its-always-drms-fault

[27] B. Sterling, "Dead Media Beat: Microsoft Plays for Sure," *Wired*, Apr. 2008. [Online]. Available: https://www.wired.com/2008/04/dead-media-be-3-2/

[28] Andreas Gal, "Reconciling Mozilla's Mission and W3C EME," May 2014. [Online]. Available: https://hacks.mozilla.org/2014/05/reconciling-mozillas-mission-and-w3c-eme