Secure Video Player for Mobile Devices Integrating a Watermarking-based Tracing Mechanism

Pablo Antón del Pino, Antoine Monsifrot, Charles Salmon-Legagneur and Gwenaël Doërr
Technicolor R&D France, 975, avenue des Champs Blancs, 35576 Cesson Sévigné, France

Keywords: Digital Right Management, Trusted Execution Environment, Secure Boot, Watermarking.

Abstract: Content protection relies on several security mechanisms: (i) encryption to prevent access to the content during transport, (ii) trusted computation environment to prevent access during decoding, and we can also add (iii) forensic watermarking to deter content re-acquisition at rendering. With the advent of next generation video and the ever increasing popularity of embedded devices for content consumption, there is a need for new content protection solutions that rely less on hardware. In this context, we propose an architecture that combines the ARM TrustZone technology, an hypervised environment built on Genode and a bit stream watermarking algorithm that inserts serialization marks on the fly in an embedded device. As a result, an attacker cannot get access to video assets in clear form and not watermarked. Reported performances measurements indicate that the induced computational overhead is reasonable.

1 INTRODUCTION

Digital Rights Management (DRM) systems protect multimedia content all along the content distribution chain. In a typical DRM infrastructure (OMA, 2008), video content is encrypted on the server side and decrypted locally on a consumer device. The distribution itself is considered to be a flawless process thanks to the security provided by cryptography. Nevertheless, at consumption time, a video player is subject to two major threats: (i) access to clear text video content (compressed or uncompressed) in memory and (ii) acquisition of rendered content e.g. with screencasting tools or camcorders.

To protect against such attacks, each DRM solution defines some specific robustness rules. Next generation video (ultra high definition, high dynamic range, ultra wide gamut) is emerging and MovieLabs, a R&D consortium sponsored by six major Hollywood studio, is pushing some requirements to harden client platforms. For instance, a key requirement is the presence of a hardware root of trust, an embedded secret that cannot be read or replaced (Microsoft, 2010; MovieLabs, 2013). This forms the basis for a chain of trust, to verify the signature and the integrity of running software, e.g. a video player. As a result, it is impossible to modify the source code to hook the video pipeline.

To prevent an external program to directly tap into memory where video may transit in clear, the media data pipeline is also required to be secure. For instance, one could provide memory scrambling protection and protection of uncompressed content to the output (HDCP) (Microsoft, 2013). Actually, MovieLabs is going even one step further and requires a secure computation environment, isolated by a hardware mechanism for critical operations, and a memory protection for this secure environment. This extends on a larger scale what Conditional Access vendors already provide in set-top boxes with hardware security modules. Such secure environment can then be used to run sensitive applications e.g. video decoding that necessarily operates on decrypted video content.

These security solutions are now well established, e.g. with TrustZone® technology largely deployed by ARM silicon partners. However, they do not address the threat of content re-acquisition after rendering. As a matter of fact, it is impossible to prevent somebody from placing a camera in front of a screen. Common practice is therefore to rely on signal processing deterrence mechanism referred to as *digital watermarking* (Cox et al., 2007). In a nutshell, video content is imperceptibly modified to convey some information and this information can be recovered even if the content has been further processed afterwards. In particular, this auxiliary communications channel can be exploited to transmit a binary code that uniquely iden-

Antón del Pino P., Monsifrot A., Salmon-Legagneur C. and Doërr G..
Secure Video Player for Mobile Devices Integrating a Watermarking-based Tracing Mechanism.
DOI: 10.5220/0005042502530260
In *Proceedings of the 11th International Conference on Security and Cryptography* (SECRYPT-2014), pages 253-260
ISBN: 978-989-758-045-1

tifies the recipient of the video. As a result, if a copy of a movie surfaces on some unauthorized distribution network, content owners can trace back to the source of the leak based on the watermark information. This strategy is routinely used today to distribute DVD/BD screeners prior to theatrical release or in digital cinema (EDCINE, 2007).

With the coming next generation video, MovieLabs promotes the use of such forensics watermark at a larger scale. "The system shall have the ability to securely forensically mark video at the server and/or client side to recover information necessary to address breaches" (MovieLabs, 2013). The objective of this paper is to propose a secure player that would combine both security mechanisms: video watermarking and decoding at the client side in a trusted environment. The proposed architecture establishes a secure media data path so that, at any time, an attacker could not get access to video assets in clear form and not watermarked.

In Section 2, we briefly review background elements that will be necessary for the proposed architecture. In Section 3, we outline the architecture of a secure video player that incorporate a watermarking-based serialization module and details a number of implementation detailed in Section 4. Performances measurements reported in Section 5 suggest that the computational overhead of our solution remains acceptable. Eventually, perspectives for future work are outlined in Section 6.

2 BACKGROUND

2.1 GlobalPlatform TEE

GlobalPlatform defines a Rich Execution Environment (REE) and a Trusted Execution Environment (TEE). The TEE is a secure area in the main processor of an embedded device whose task is to store, process, and protect sensitive data. In particular, the TEE is expected to resist to all known remote and software attacks, and a set of external hardware attacks. Any code running inside the TEE is trusted in authenticity and integrity, thereby protecting assets and code from unauthorized tracing and control through debug and test features.

Client applications running in the REE make use of a TEE client API to communicate with their associated trusted application (GlobalPlatform, 2010). A salient feature of the model is the ability to communicate large amounts of data using a memory area accessible to both the TEE and the REE, thus avoiding unnecessary copies of data. In our development, the TEE Client API and Communication Stack is based on the Open Virtualization TEE solution (Sierraware, 2012).

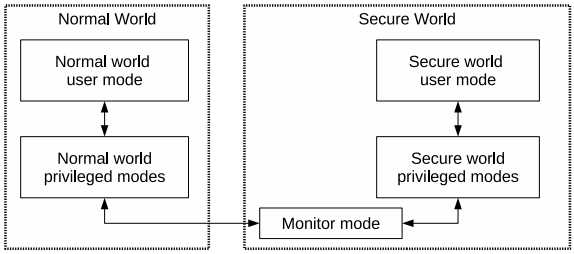

Figure 1: TrustZone operation modes

2.2 Secure Boot

GlobalPlatform specifies that the TEE must be instantiated through a secure boot process using assets bound to the system on chip (SoC) and isolated from the REE. The role of a secure boot is to implement a chain of trust (Arbaugh et al., 1997). The ROM of the SoC is usually the only component in the system that cannot be trivially modified or replaced by simple reprogramming attacks. As a result, it routinely hosts the first stage boot-loader that acts as the root of trust. Subsequent components are then authenticated before being executed. In this study, we will use the Freescale i.MX53 low cost SoC, that provides a multi-stage secure boot process based on the high assurance boot library (HAB).

2.3 TrustZone

The implementation of a TEE depends on the platform architecture. The i.MX53 is equipped with the ARM TrustZone technology, which offers a hardware solution to obtain a rich open operating environment with a robust security solution (ARM, 2009). In TrustZone terminology, the domains for REE and TEE environment are referred to as *Normal World* and *Secure World* respectively. The security of the system is achieved by partitioning all of the SoCs hardware and software resources so that they exist in one of the two worlds – the Secure World for the security subsystem, and the Normal World for everything else.

For instance, two virtual cores are associated to each physical processor core. A mechanism known as *Monitor Mode* then allows to robustly context switch between worlds, as illustrated in Figure 1. Triggering the context switch requires the execution of a specific processor instruction that only runs in privilege modes. The execution of a Secure Monitor Call (SMC) instruction causes a processor exception that is handled by a routine defined on the Monitor exception table.

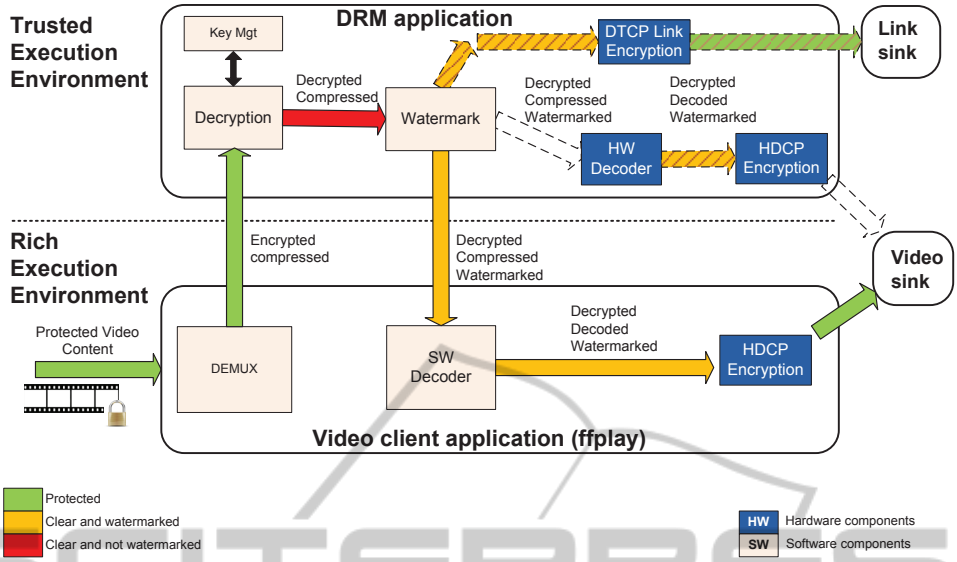

Figure 2: Secure data path in the proposed architecture. Video content is never accessible in clear and without a forensic watermark. Performing watermarking at the bit stream level offers the possibility to shift the decoding process in the non-trusted environment and thereby lighten the load of the TEE.

2.4 TrustZone VMM

Managing both environments requires a software layer that supports two features: a kernel to run applications in the trusted environment and a virtual machine monitor (VMM) to switch between environments. In order to reduce the attack surface, μ-kernels are usually preferred. Several commercial solutions based on TrustZone security extension instruction set exist: Trusted Foundations from Trusted Logic Mobility in Tegra-powered tablets, Mobicore™ from Giesecke & Devrient in the Galaxy SIII, and more recently <t™-base-300, from the joint-venture Trustonic. On the open source side, a good alternative would be L4Android (Lange et al., 2011) but it does not manage TrustZone security extension. In contrast, Genode is an operating system framework that implements a VMM on top of the TrustZone hardware and that could be based on various μ-kernels, e.g. L4, Fiasco and Genode. Since version 13.11, Genode's hardware-based μ-kernel has been supporting Freescale i.MX53 with TrustZone and we therefore adopted this full Genode setup.

3 SECURE VIDEO PLAYER

Figure 2 depicts the overall architecture for the secure player that we are proposing. The video client application runs in a regular Linux environment, and is extended with some secure components within a TEE. Following the principle of isolation for ARM TrustZone™, the most sensitive tasks (key management, decryption and watermarking) are performed inside the TEE. Since they run in a confined memory space which is not physically accessible from the regular environment, capture of decrypted and non-watermarked content is prevented.

3.1 DRM Module in TrustZone

TrustZone architecture differs from traditional protection in set-top boxes. Instead of encapsulating security critical processes in extra dedicated chips, the main processor is enhanced by the addition of a new secure execution mode and a specific memory controller for memory partitioning. An important by-product is that key management can now be performed in software in the TEE, which significantly facilitates the design of new products that would require otherwise the collaboration from chip vendors.

The video-on demand player Video Hub on Samsung's Galaxy SIII, developed by Discretix, provides a good illustration of how TEE may be used for content protection. Several trusted components, also referred to as *trustlets*, embeds DRM modules such as PlayReady® or Widevine® inside the Mobicore framework. Discretix's secure PlayReady DRM solution (Discretix, 2011) includes the truslet itself, but also a secure user interface, a secure storage, access to cryptographic engines and hardware codecs. In other words, it implements a secure media pipeline totally

enclosed into the TEE secure world.

The number of features handled in the secure world is significant and one may wonder if it may not yield any vulnerability. A software level analysis of TrustZone OS and trustlets (Behrang, 2013) actually revealed an issue that could allow unauthorized communication between low privileged Android processes and Mobicore kernel. Now, even if such integration vulnerabilities were fixed, there would still be the risk of unauthorized re-acquisition. Content decoded in the TEE is either forwarded to another devices through a protected link or to the rendering engine. Security flaw or not, content will eventually be left unprotected, and thus vulnerable, at some point to be presented in clear to human beings.

3.2 Bit Stream Watermarking

Digital watermarking is the technology of choice to deter content re-acquisition. This being said, most video watermarking algorithms feature too much complexity to be able to watermark content on the fly on resources-constrained mobile devices. Still, a new breed of watermarking systems recently emerged that separate the watermarking process in two elementary steps (Zou and Bloom, 2008; Zou and Bloom, 2010).

The main objective is to make the watermark embedding module as simple as possible for efficiency. This calls for watermarking systems that operate prior to decompression, e.g. by swapping a few bytes at different locations directly at the bit stream level. However, random alterations of the bit stream is prone to significantly impair the quality of rendered video. It is therefore necessary to perform a computationally intensive profiling of the original bit stream to identify locations that could be modified as well as the associated replacement bytes that could be used without introducing noticeable visual degradation. A key feature is that this profiling operation only needs to be performed once, regardless of the number of recipients whom the video will be sent to.

While the proposed architecture is applicable for any bit stream watermarking system, we will focus in the remainder of the article on an algorithm that operates directly in the H.264 AVC/CABAC bit stream (Robert and Doërr, 2013). The profiling module essentially generates a sequence of instructions that will be consumed by the embedding module. Most instructions can be viewed as a triplet containing (i) an offset where the modification can be made, (ii) a 2-bytes word to use to embed a ‘0’, and (iii) a 2-bytes word to embed a ‘1’. To avoid synchronization problems, these instructions are finally interleaved at the H.264 signaling level. An H.264 bit stream is essentially composed of a network abstraction layer unit (NALU). If a NALU is found to host watermarking locations, it is preceded by a NALU whose header indicates that it contains supplemental enhancement information (SEI), namely the instructions required to watermark the next NALU.

As a result, the embedding module operates NALU by NALU. If the incoming NALU is a SEI containing watermarking instructions, the payload of the NALU is stored in memory. If it is a NALU containing video information, the swaps necessary to embed the bits of the locally stored identifier are performed using the instructions in memory (if there is any) and the memory is reset afterwards. The local identifier is loaded at startup in a secure way i.e. it is impossible to modify the identifier that will be subsequently embedded in videos. This identifier can point to a user/device whose privileges need to be revoked or the built of a software that needs to be updated. In some sense, this watermarking system is very similar to `SequenceKey` proposed by IBM for optical disks protection (Jin et al., 2004), although it operates at a much finer granularity.

3.3 Analysis

Figure 2 depicts two alternate media data path implementations. The first one embeds the full video pipeline, from decryption to rendering, inside the TEE. In the second one, only the critical path, from decryption to watermarking is included into the TEE. The forensic watermarking provides a tracing mechanism in case of leak and subsequent operations, from decoding to output protection, can be done ‘safely’ in the REE.

The first solution avoids additional memory transfers and the clear content, compressed or uncompressed, is never accessible from the REE. From a security and performance perspective, it looks like the perfect solution. However, it also comes with some inherent drawbacks. The large number of system features in the TEE (secure video driver, decoding, sink encryption) significantly increases the amount of code to secure, which exposes to more risks and potential maintainability issues.

In contrast, the second solution offers a good trade-off between a good level of security and an open customizable architecture. The main objective is to keep the DRM module running in the TEE minimal and non CPU intensive. As such, the chosen bit stream watermarking technology is perfectly in line with this strategy. Moreover, since the decryption and watermarking tasks do not require decoding video content, the proposed DRM module is not tied to a specific hardware or software codec implementation or to a system component. The clear separation

of the security features grants the flexibility to shift all application features, such as decoding and rendering, in the normal world. Software maintenance is thereby greatly facilitated. For instance, the video rendering module, subject to bug fixes and upgrades, can be easily updated while the DRM module remains stable and unchanged.

Another advantage of introducing a watermarking-based tracing mechanism in a secure player is related to the protection of the output sink that is routinely found to be the weakest link of the system. For instance, numerous attacks against high definition copy protection (HDCP) have been publicized in 2010 due to software key discovery, lack of diversity and broken root of trusts (Rosenblatt, 2011). Watermarking does not prevent copyright material to be pirated through a broken link but at least provides a tracing mechanism to identify the source of the leak. It exemplifies the virtue of having multiple lines of defense in order to avoid single points of failure.

4 IMPLEMENTATION DETAILS

Figure 3 depicts the interactions between the different software modules that we have implemented to instantiate the video player design proposed in Section 3. It closely follows the generic architecture advocated by Global Platform. On the left hand side, the REE stands for a Linux Kernel and a `rootfs` that offer a complete execution environment for user applications. On the right hand side, the TEE is composed of the Genode API and two Genode user applications (VMM-TEE and DRM Trusted Application). In between, a low-level piece of software is in charge of several hardware-dependent tasks, including in particular communication processes between the non-secure and secure domains.

4.1 TEE Modules

Genode's μ-kernel supports TrustZone and can manage context switches between worlds from the application Virtual Machine Monitor (VMM). VMM handles all aspects related to the para-virtualization of the REE (memory management, exception handling, secure access to hardware, etc). We complemented this application with a module called Genode TEE that is in charge of the interactions between the TEE Client in the normal world and the trusted application (DRM Trusted Application in our case) in the secure world. The resulting joint-application is VMM TEE.

The VMM has two main duties. First, it is in charge of configuring the platform to load and run the REE Linux kernel. It sets up the memory regions where the kernel image will be loaded, configures the security properties to keep the TEE isolated from the REE, launches the Linux kernel image, and gives it the control of the machine. The second task of the VMM is to listen for context switch requests emitted by the TEE Client Driver. Whenever they occur, it looks up some parameter values and decides whether to transfer control or not to Genode TEE handler function.

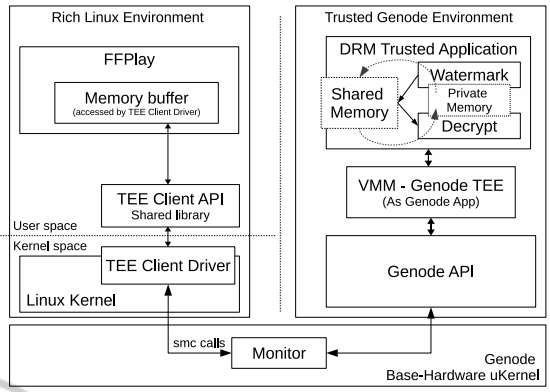

Figure 3: Software architecture of the proposed secure video player.

On its side, Genode TEE serves as an interface between the TrustZone hardware and the trusted application running in the TEE. In particular, it processes a number of commands, e.g. open session with trusted application, send command to trusted application, close session with trusted application, etc. In addition to such default command, each trusted application can define its own set of command. In the case of our DRM trusted application, we defined three commands as well as their associated behavior:

1. `TEE_DRM_SET_LICENCE` command is sent when the secure player attempt accessing protected content. It is in charge of extracting relevant information securely stored in the associated content license, e.g. content decryption keys and usage rights.
2. `TEE_DRM_INIT_WM` command is issued once when the secure player is launched to initialize resources allocated to the watermarking module. In particular, it allocates some buffers to temporarily store watermarking instructions and securely load the serialization payload that will be embedded by the watermarking routine.
3. `TEE_DRM_DECWM_PACKET` is a recurring command that triggers the decryption and watermarking of a video packet. The pointer to the region in the shared memory where the video packet is accessi-

ble must be sent in parameter.

For the sake of clarity, the interactions between the REE and the TEE will be further detailed in the next subsection with the workflow of our secure player.

Security provided by the DRM trusted application is twofold. The content decryption key is retrieved from the associated content license, which are encrypted using a platform/device-dependent public key. The corresponding private key, which will be used to decrypt the license and thereby get access to the content decryption key, is installed by the manufacturer in each consumer board and is only accessible in the TEE. Thanks to TrustZone's isolation, it guarantees that crypto-credentials will never leave the secure world. The second security mechanism comes from the memory management. The DRM trusted applications reads encrypted video packets from the shared memory but stores the decrypted packet in a private memory that is unaccessible from the REE. The application pushes the decrypted content back into the shared memory, only after it has been watermarked. Thus, unencrypted content without watermark is never accessible from the non-secure domain.

4.2 Secure Video Player Workflow

In our implementation, we are using the FFPlay application[1], a portable media player combining FFmpeg and SDL libraries[2]. Due to its simplicity, establishing the communication with the TEE Client API shared library is reasonably easy.

When FFPlay is launched, it issues a command `TEEC_InitializeContext` that prepares driver structures and buffers, i.e. it initializes the TEE. Immediately afterwards, a command `EEC_OpenSession` creates a session with the DRM Trusted Application located in the TEE. Everything is fairly standard so far and would be the same for any trusted application. From this point onward, though, the interactions become application specific and are triggered with the `TEEC_InvokeCommand` function, that can specify any application command which has been defined.

To set up the watermarking environment, FFPlay first invokes the command `TEE_DRM_INIT_WM`. On the TEE side, the DRM Trusted Application allocates a number of buffers, e.g. to temporarily store watermark embedding instructions, and places in some secure memory the serialization identifier that will be embedded by the watermarking module. When the VMM handles back control to FFPlay, the watermarking module of the DRM Trusted Application is ready to process any incoming video packet. Still, it can only handle decrypted content. When FFPlay attempts playing protected content, it therefore invokes a command `TEE_DRM_SET_LICENCE` to tell the DRM Trusted application to use the platform/device key to recover the content decryption keys from the license that is passed in argument.

Table 1: Percentage of processing time devoted to security tasks along the video processing pipeline of the proposed secure player for different setups and H.264 profiles.

	Decryption			
	REE		TEE	
	Main	*High*	*Main*	*High*
No watermark	0.04	1.03	2.25	5.87
Watermark	N/A	N/A	2.61	6.83

Equipped with content decryption keys and a watermarking module ready to go in the TEE, FFPlay can enter its nominal regime. It reads protected video packets from the video file and issue a command `TEE_DRM_DECWM_PACKET` to the DRM Trusted Application. On the TEE side, the decryption module first decrypts the video packet accessible in the shared memory and stores the result in private memory. For simplicity, in our implementation, a video packet accounts for an integer number of H.264 NALUs. In the general case, one would need either to introduce a new module in the TEE to parse video packets and forward full NALUs to the watermarking module, or to interleave watermarking instructions at the container level so that the watermarking module can operate packet by packet instead of NALU by NALU.

When the watermarking module kicks in, it inspects the header of the decrypted H.264 NALU. If it is a SEI containing watermarking instructions, the payload of the NALU is copied in a dedicated buffer and the NALU itself is filled with dummy bytes to prevent access to watermarking instructions outside the TEE. If it is a regular NALU, it consumes the instructions (if there are any) stored in the buffer to watermark the NALU located in the private memory. Once the video packet (containing one or several NALUs) is fully processed, the resulting decrypted and watermarked packet located in private memory is copied back in the shared memory at the exact same location that it comes from. When the VMM gives the control back to FFPlay, it can forward the processed video packet further along the video pipeline, e.g. video decoding and rendering.

When FFPlay is closed, it invokes a command `TEEC_CloseSession` to properly release resources on the TEE side.

[1] http://www.ffmpeg.org/

[2] http://www.libsdl.org/

5 PERFORMANCES MEASUREMENTS

To evaluate the processing overhead induced by the proposed architecture, we conducted a number measurement campaigns with movies in high definition. For completeness, the videos were compressed with two different H.264 profiles that translate in various bit rates and average NALU sizes. For instance, the encoder bit rate is much higher for the high profile than for the main profile. In our case, this translates in NALUs being 27 times larger, in average, for the high profile.

For each movie, we measured the percentage of processing time dedicated to security tasks with respect to the total processing time during playback. All time measurements have been done using the function `gettimeofday()`. Such measurements have been performed for various placements and setup of the decryption and watermarking modules and the average results are reported in Table 1.

The upper left corner of the table reports performances when the video player is fully implemented in the REE realm without using TrustZone. In this case, security features reduce to decryption only and the first observation is that decryption processing task is negligible with respect to decoding. The increase from 0.04% to 1.03% between the main and high profiles accounts for the increase in size of the processed NALUs. Decryption time is strictly proportional to the NALU size, whereas decoding time is not.

The upper right corner of Table 1 showcases the impact of shifting the decryption module inside the TEE. It introduces significant overhead that is mostly due to some limitations in the current implementation of the TEE Client Driver. First, the shared memory is not fully operational and we actually had to copy encrypted video packets into the TEE private memory instead of simply reading from the shared memory. Moreover, a memory copy from the REE to the TEE actually requires two memory copies: from the user space to the kernel space and from the kernel space to the DRM Trusted Application (or the other around on the way back). All in all, it induces a total of four memory copies for a decrypt-watermark loop of a single video packet. Still, should these limitations be fixed in the future, we could fully leverage TrustZone's shared memory. A priori, the processing time overhead could be reduced to around 0.6% for the Main profile and 2.25% for the High profile.

Finally, the lower right corner of the table reports on the impact of introducing the proposed watermarking module in the architecture. For both profiles, the processing time overhead remains below 1% wich clearly highlight the minimal footprint of this module. Again, the overhead is larger for the high profile. This is due to the fact that larger NALUs offer more potential embedding sites that translates in byte swaps for the embedding module. This being said, the overhead of the watermarking process can easily be limited if necessary by discarding embedding changes when formating the watermarking instructions at the profiling stage. The flip side of the coin is that it will require more video time to embed a full identifier.

From the consumer's perspective, the addition of the security modules is transparent i.e. the media consumption experience is the same. We did notice some occasional freeze of the display but they were actually due to FFPlay which was not always able to maintain real-time decoding for high-definition content on the i.MX53. This could be solved in the future by using a more efficient software player or even a dedicated video decoding chip.

6 CONCLUSION AND FUTURE WORK

Forensic watermarking is currently used in professional environments. However, with the advent of next generation entertainment content, there is momentum building to deploy such traitor tracing mechanism at a larger scale. Content serialization could either be done on the server side or on the client side. Solutions on the server side come with their own shortcomings e.g. storage overhead and reduced watermark granularity if video segments are precomputed (Jarnikov and Doumen, 2011) or computational and bandwidth overhead if segments are watermarked on the fly. On the other hand, solutions on the client side most often rely on hardware to host their security features (PRIMA Cinema, 2013).

In this paper, we intended to study an alternate paradigm to design secure video players that would rely on a software architecture in order to be possibly embedded onto cheap consumers electronics devices. It essentially leverages on two recent developments: (i) GlobalPlatform and its ARM TrustZone incarnation that provides a trusted execution environment where sensitive processes can be run safely, and (ii) 2-step bit stream watermarking that allows seamless integration along the video processing pipeline. By combining these two elements, the attacker never has access to clear video content that is not watermarked with some serialization identifier.

Our proof of concept using the i.MX53 board, Genode's environment and FFPlay video player clearly demonstrated the feasibility of the proposed

solution. While the processing time overhead may first look significant, it comes for a large part from suboptimal design choices that are likely to be fixed in the future but that induce costly memory copies at the moment. Moreover, the consumer experience is more affected by FFPlay that struggles to keep the real-time cadence for high-definition content. In the future, we intend to replace FFPlay by a more efficient player that could exploit hardware acceleration, such as gstreamer-based player Totem.

Another avenue for future work is to lift the simplifying assumption that an encrypted video packet contains an integer number of NALUs. A first solution is to insert an additional interface between the decryption and watermarking modules, that would assemble NALUs from incoming decrypted video packets. An alternate, possibly more efficient, solution would be to raise the transmission of the watermarking instructions at the container level rather than the video essence level. As a result, the watermark embedding process, a simple byte swapping engine, could operate video packet by video packet.

REFERENCES

Arbaugh, W. A., Farber, D. J., and Smith, J. M. (1997). A secure and reliable bootstrap architecture. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 65–71.

ARM (2009). ARM security technology – Building a secure system using TrustZone technology. Technical report.

Behrang (2013). A software level analysis of TrustZone OS and trustlets in Samsung Galaxy phone.

Cox, I. J., Miller, M. L., Bloom, J. A., Fridrich, J., and Kalker, T. (2007). *Digital Watermarking and Steganography*. 2nd edition.

Discretix (2011). DRM, Ensuring secure content protection solutions for next generation consumer device. Technical report.

EDCINE (2007). *European Digital Cinema Security White Book*.

GlobalPlatform (2010). GlobalPlatform device technology – TEE client API specification. Technical report.

Jarnikov, D. and Doumen, J. M. (2011). Watermarking for adaptive streaming protocols. In *Proceedings of Secure Data Management*, volume 6933 of *LNCS*, pages 101–113.

Jin, H., Lotspiech, J., and Nusser, S. (2004). Traitor tracing for prerecorded and recordable media. In *Proceedings of the 4th ACM Workshop on Digital Rights Management*, pages 83–90.

Lange, M., Liebergeld, S., Lackorzynski, A., Warg, A., and Peter, M. (2011). L4Android: A generic operating system framework for secure smartphones. In *Proceedings of the ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*, pages 39–50.

Microsoft (2010). Robustness rules for PlayReady final products. Technical report.

Microsoft (2013). Compliance rules for PlayReady final products. Technical report.

MovieLabs (2013). MovieLabs specifications for next generation of video and enhanced content protection. Technical report.

OMA (2008). DRM architecture. Technical report.

PRIMA Cinema (2013). http://primacinema.com.

Robert, A. and Doërr, G. (2013). Impact of content mastering on the throughput of a bit stream video watermarking system. In *Proceedings of the IEEE International Conference on Image Processing*, pages 4532–4535.

Rosenblatt, B. (2011). The new technologies for pay TV content security. Technical report.

Sierraware (2012). Open virtualization for TrustZone overview.

Zou, D. and Bloom, J. A. (2008). H.264/AVC stream replacement technique for video watermarking. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 1749–1752.

Zou, D. and Bloom, J. A. (2010). H.264 stream replacement watermarking with CABAC encoding. In *Proceedings of the IEEE International Conference on Multimedia and Expo*, pages 117–121.