Securing Video on Demand Content with SGX: A Decryption Performance Evaluation in Client-Side

Ricardo de S. Costa[1], **Daniel F. Pigatto**[1], **Keiko V. O. Fonseca**[1], **Marcelo de O. Rosa**[1]

[1]Pós-Graduação em Engenharia Elétrica e Informática Industrial (CPGEI)
Universidade Tecnológica Federal do Paraná (UTFPR)
80230-901 – Curitiba – PR – Brazil

`rcosta.am@gmail.com, {pigatto,keiko,mrosa}@utfpr.edu.br`

Abstract. *Video on Demand (VoD) is presently a bandwidth demanding application of IPTV networks but also a source of revenue for content providers, broadcasters and infrastructure providers. Customers can select TV content like movies and shows from a wide digital media library, and are no longer tied to a fixed schedule. The programming content is digitally compressed, stored and kept at VoD storage drives and flows from them to residential set-top boxes (STB) according to the user demand. However VoD service should set the streaming of a chosen content only to authorized clients, that is, the system should secure the data transfer and storage in order to prevent copyright infringements or unfair business competition with content illegally acquired. This paper proposes the adoption of SGX technology as a solution for a secure VoD service focusing in client-side (embedded devices) and evaluating its decryption performance. We show that video chunks normally transfered between media servers and real STBs can be decrypted in approximately* $150\mu s$*, on average, which is enough for using this technology in STBs.*

Resumo. *O Video on Demand (VoD) é atualmente uma aplicação que exige uma alta largura de banda em redes de IPTV, mas também uma fonte de receita para provedores de conteúdo, emissoras e provedores de infra-estrutura. Os clientes podem selecionar conteúdo de TV, como filmes e programas, a partir de uma ampla biblioteca de mídia digital, e não estão mais vinculados a um cronograma fixo. O conteúdo da programação é comprimido digitalmente, armazenado e mantido em unidades de disco VoD e fluxos a partir deles para set-top boxes (STB) residenciais de acordo com a demanda do usuário. No entanto, o serviço de VoD deve definir o streaming de um conteúdo escolhido apenas para clientes autorizados, ou seja, o sistema deve proteger a transferência de dados e armazenamento, a fim de evitar violações de direitos autorais ou concorrência empresarial injusta com conteúdo adquirido ilegalmente. Este artigo propõe a adoção da tecnologia SGX como uma solução para um serviço de VoD seguro com foco no cliente (dispositivos embarcados) e avaliação do desempenho de descriptografia. Nós mostramos que blocos de vídeos normalmente transferidos entre servidores de mídia e STBs reais podem ser decifrados em aproximadamente* $150\mu s$*, em média, que é suficiente para usar essa tecnologia em STBs.*

1. Introduction

Presently, telecommunications companies providing TV services are looking for new solutions that link TV content distribution to other interactive services like on-line shopping, games and movie rentals, as convenience features for their users. One of these services involves combining video rental business and the use of recording capabilities of Set-top Boxes (STBs) in order to allow customers to select and view available movies at any time, a service usually known as Video on Demand (VoD) [Peltoniemi 1995].

VoD traffic is expected to double while consumer Internet video traffic will globally reach 79% of all consumer Internet data traffic by 2018 [CISCO 2017]. As VoD applications grow, also grow security issues related to digital rights infringements or non-authorized access to protected video contents [Akhyar et al. 2015]. Recently, FCC has changed rules [FCC 2016] aiming at rules that will both empower consumers to choose how they wish to access their subscribed multi-channel video programming, and promote innovation in displaying, selecting, and using it and other video programming available to consumers by different video content and service providers. These new rules should increase competition and affect the way the providers implement protection to video content.

On the other hand, hardware-based security technologies have been developed to be available in device processors as a security co-processor in order to face the increasing number of cyber attacks that exploits both software and hardware vulnerabilities. However, the migration of legacy VoD software to these new secure platforms requires huge efforts of software planning and design to enforce protection of video content.

In this paper, we claim that hardware-based security technologies [Intel 2016a, ARM 2016] can be used to provide a secure execution environment that protects both code and data software as a MaaS (Metal-As-A-Service) to secure the video content, the video management software, and its private data inside STBs. Moreover, the MaaS approach should provide a safe computing environment (in our work, an enclave) in which software can run free from external observation and modification of its code and private data [Lal and Pappachan 2013], opening new perspectives for cloud-based secure VoD applications [SecureCloud 2016]. Our focus is to determine how feasible is the use of enclaves to enforce Digital Rights Management (DRM) restrictions at the client-side of video systems.

The paper is structured as follows: Section 2 provides an overview of VoD concepts; Section 3 shows how HTTP-based Adaptive Streaming works with HTTP Live Streaming (HLS) technology; Section 4 presents Digital Rights Management (DRM) concepts; Section 5 explains cryptography concepts needed to understand our contribution; Section 6 reviews the hardware security solution used here (Intel Software Guard Extensions or SGX); Section 7 explains our proposed solution using a Trusted Execution Environment (TEE) [Microsoft 2017] implementation like SGX applied to a VoD System with decrypting performance evaluation at client-side embedded devices (also STBs); Section 8 presents the performance evaluation results and discussions; Section 9 brings our conclusions, including possible future works.

2. Video-on-demand (VoD)

Video on Demand (VoD) service allows users to select and watch video contents. Such services should allow controls to forward, backward, hold and stop the video streaming up to an expiration date based on a service acquisition contract. The basic concept of VoD is to store channels/contents and forward them to the user. The video/data content can be kept on a central storage and retrieve (media) server that streams content simultaneously to hundreds of users, or distributed to several media servers along the network and then be shared among the users [Irawan 2013].

Each connection to a VoD system requires a bi-directional communication between client and server, and each server keeps a set of available videos to all users. The server processes users requests trying to answer all them as fast as possible. Hundreds or thousands different clients can simultaneously require access to movie catalogs or watch videos. The VoD system should provide a specified Quality of Service (QoS) level to its video streaming sessions [Liebeherr 1995].

3. HTTP-based adaptive streaming (HAS)

The storage volume of a media server and the network traffic for VoD services are continuously increasing. The number of videos stored in a media server has dramatically increased along with the growth of social media and mobile service and video streams need to be provided at different qualities (different bit-rates) as the types and capabilities of client devices become more diverse [R. Mohan and Li 1999]. For seamless streaming service, studies [Stockhammer 2011, Müller et al. 2012] have suggested adjusting network traffic by selecting the quality of a video stream according to the network conditions, optimizing content delivery network (CDN) distribution infrastructures. HTTP-based adaptive streaming (HAS) solutions that adapt the quality of the video stream based upon the consumer's changing bandwidth include MPEG's Dynamic Adaptive Streaming over HTTP (DASH), Apple HLS, Adobe HTTP Dynamic Streaming (HDS) and Microsoft Smooth Streaming. In particular, HLS is widely supported in various media server providers such as Akamai, Amazon CloudFront and Adobe Media Server (AMS), and players such as VLC media player and hand-held device OSs (Android, IOS, Windows, and Tizen) [Hur et al. 2017].

3.1. HTTP Live Streaming (HLS)

HAS is an adaptive streaming technology based on HTTP. It allows the client to choose an appropriate quality version for each video segment based on the available bandwidth between the server and client. HLS is one of the widely known HAS solutions developed by Apple [Apple 2016], thus we use HLS in this work as a general HAS solution. The HLS server divides a video into segments encoded at a variety of data rates and stores them in a storage unit. The client downloads video segments from the server via HTTP. Compared to traditional streaming technologies, HLS advantages are [Chakraborty et al. 2015]:

- It is capable of traversing any firewall or proxy server that lets through standard HTTP traffic.
- It allows video fragments to be cached by proxies, thus reducing the load on the source server and improving access speed for download.

- Upon receiving a video request from the mobile client, the web server checks the manifest files containing the meta-data for each encoded streams and sends segments of video data at an appropriate bit-rate level to the client.

For video playbacking, the client requests video segments set to a quality suitable to its network conditions by referring to the server manifest files. Then it plays back the segment downloaded from the server. The client measures the transmission time of the requested segment during download, and discovers current network conditions based on the measured time and the volume of the segment downloaded [Chakraborty et al. 2015], automatically requesting new video segments with suitable quality.

4. Digital rights management (DRM)

Digital rights management defines methods and technologies to control the access of digital content owned by publishers, copyright holders, and/or individuals. Only authorized users have access to such a digital content and the digital content is protected against unauthorized replication or broadcasting. Managing the access to digital media is important for companies concerned with copyrights, particularly entertaining industries, who are partly or wholly dependent on the revenue generated from their work [encoding.com 2016]. The two usual methods to impose DRM on media data are:

- Use restrictive licensing agreement, where the access to copyrighted material is granted according to specify legal agreements between the material owners and subscribers before the latter can get access to that material. Even public domain software impose some legal agreement over the supplied digital material;
- Encrypting, scrambling, and/or embedding a tag, where the digital material is modified in order to block attempts to copy it or distribute it to unauthorized users, and to control the access to that material.

Particularly the latter method involves applying cryptography which can be based on software and/or hardware.

5. Principles of cryptography and VoD service

Within a VoD environment it would be insecure to deploy only symmetric keys for all content since the keys would have to be often renewed and sent over the same network used to deliver the content. Intruders intercepting the client-server communication would easily capture the keys and be able to have access to the encrypted content. Processing resources required for real-time applications using asymmetric keys can impact VoD service costs.

Generally both symmetric and asymmetric cyphering are combined within VoD environments: first, a STB and a media server transfer to each other public keys and negotiate the requested video content, along with additional STB information. Next, the media server transfers a random symmetric/secret key used to cipher video segments. Therefore video content can be securely and fast transferred from a media server to a specific STB according to their privileges and requests. This procedure is similar to those of secure Internet protocols like SSH and HTTPS.

6. Software Guard Extensions and Enclaves

Intel Software Guard Extensions (SGX) [Knight 2015] is a set of x86-64 ISA extensions and an opt-in feature enabled by BIOS that allows to set up protected execution environments (called enclaves) without requiring trust in anything but the processor and the code users place inside their enclaves [Anati et al. 2013]. SGX reduces the attack surface by prohibiting access of higher privileged software such as operating system, kernel drivers, hypervisors, and system management mode (SMM) handlers among others. Additionally, enclaves are encrypted while resident in system memory which protects them against several attacks like memory scrapping/scanning and physical attacks. SGX works by providing multiple protection borders around sensitive code and/or data. A high-level overview of the SGX architecture is shown in Figure 1.

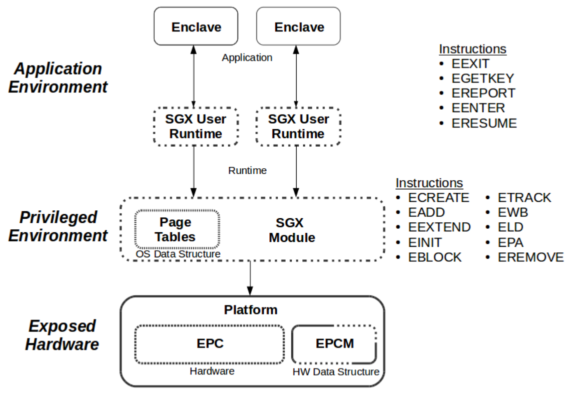

Figure 1. SGX high-level SW/HW architecture [Intel 2016b]

When loading an enclave into memory, CPU measures its content in a chained cryptographic hash log and stores it in a register called `MRENCLAVE`, similar to Platform Configuration Register (PCR) on Trusted Platform Modules. Before running an enclave, CPU verifies the content of `MRENCLAVE` against a vendor-signed version and aborts on a mismatch. Hence, CPU guarantees for integrity of the enclave startup [TCG 2014].

In order to assess enclave security, SGX provides attestation mechanisms [Anati et al. 2013]. Local attestation allows an enclave to verify if another enclave runs on the same physical CPU. Remote attestation can be used by a remote party to check if an enclave is indeed running on a genuine Intel CPU. It also allows initial provisioning of keys and secrets to set communication between enclaves: this is required since enclave code is public, not allowing to embed secrets directly into its binary code. Attestation is based on a CPU-generated signed report structure which is able to hold additional user data (including keys and secrets as mentioned before).

SGX also allows an enclave to obtain a sealing key bounded to the local CPU and to the creator of the enclave. Therefore the same sealing key can only be queried from exactly the same enclave, if loaded correctly on the same, genuine Intel CPU. The enclave can use the sealing key to encrypt and decrypt arbitrary data for off-line storage, preserving data across multiple executions of a given enclave [Intel 2016b].

During enclave initialization, the CPU verifies enclave `EINITTOKEN` (several attributes to enforce, including the debug mode flag). It has to be signed by a special launch key, which is owned by launch enclaves that are issued by Intel. Hence, by issuing proper launch enclaves, Intel controls which enclaves are to be executed in debug or production mode. The SGX evaluation SDK is shipped with a launch enclave for debug enclaves only [Intel 2017] while enclaves for production mode requires proper licenses from Intel [Johnson et al. 2016].

7. Methodology

This section details the methodology applied in our work: the system design of our work, the procedures for performance evaluation, and details of our implementation (including video decryption enclave solution and the use of measurement time functions for performance evaluation, hardware and software setups, and description of the video samples used for tests).

7.1. System design

Generically, a STB needs to retrieve encrypted video chunks (data blocks of video) from a media server, decrypts them inside enclaves, and finally playback them. Considering that SGX paradigm imposes additional delays to this procedure due to its need to transfer data from/to enclaves and to encrypt enclave memory, we evaluated the feasibility of using SGX-based STBs for VoD applications. Particularly, since the video is broken down into data chunks when transfered from a media server to a STB, we evaluate different sizes of data chunks in order to determine the optimal size for the VoD process.

Therefore we focused on evaluating the decryption performance inside an enclave. Figure 2 shows a basic sequence diagram between VoD server and a STB player (items highlighted in red emphasize our research focus).

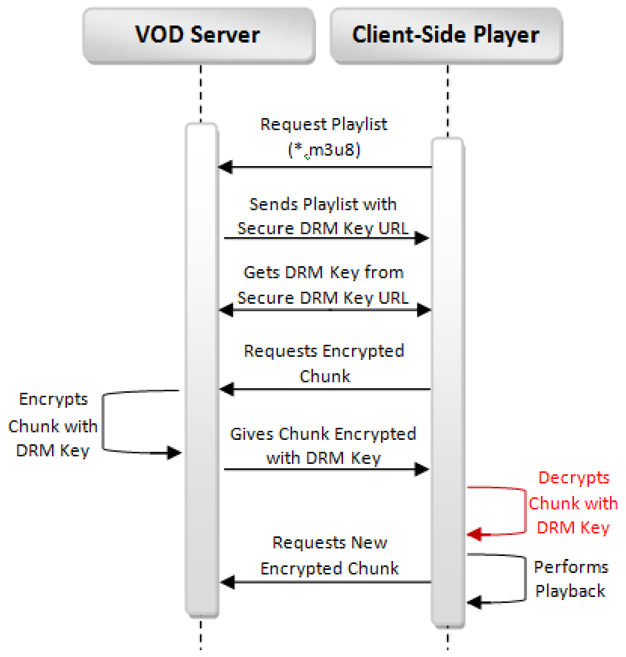
VOD Server
Client-Side Player
Request Playlist (*.m3u8)
Sends Playlist with Secure DRM Key URL
Gets DRM Key from Secure DRM Key URL
Requests Encrypted Chunk
Encrypts Chunk with DRM Key
Gives Chunk Encrypted with DRM Key
Decrypts Chunk with DRM Key
Requests New Encrypted Chunk
Performs Playback

Figure 2. Basic VoD client-server sequence diagram communication (our proposal focus on the performance of red items)

7.2. Environment setup

Considering that a STB is currently an embedded system, we used 2 computer configurations to evaluate the video decryption performance (both supporting SGX): The first consisted on a personal computer (PC) with an Intel® Xeon E3-1280 v6 4-core 3.9 GHz processor and 24 GB of RAM. The last configuration consisted on a notebook Dell Latitude 5000 Series (5480) with an Intel® Core I7-7600U v6 4-core 2.6 GHz processor and 16 GB of RAM. The Enclave Memory Size was set to 128 MB directly on BIOS configuration. All computer systems run Linux operating systems (Ubuntu version 16.04 with Kernel version 4.4.0-121-generic) and Intel® SGX SDK version 2.0.40950. Our software was generated by GCC Toolchain version 5.4.0.

7.3. Video samples, chunks and input buffer sizes

We used videos samples listed in Table 1 encoded in Common Intermediate Format (CIF) format and with time duration of 10 seconds. Following current VoD procedures, we used `ffmpeg` to break all videos in 2-second chunks (comprising video fragments) resulting a total of 5 chunks of videos. This size was chosen as an optimal segment size for a network setting of real scenarios with web servers/CDNs and HTTP 1.1 persistent connections also including live video transmission [Lederer 2015]. Along with these chunks we produced an index configuration file (`m3u8` extension) for each video (in real applications all files are transported from server to STBs over private networks). Despite the fact that HLS supports several alternate video resolutions that can be defined on an index configuration file, in this paper we consider a single resolution only.

The video chunks were re-encoded to H.264 with Baseline Profile, designed primarily for lower-cost applications with limited computing resources and widely used in videoconferencing and mobile applications [Huang 2008]. A STB with SGX should decrypt an entire chunk below 4 seconds [Lederer 2015], in order to meet reliable QoS.

Although the size of the video chunks are fixed in time, these chunks have different size in bytes. Such variation is directly related to the video content (mainly inter- and intra-frames distribution of color pixels).

Fixed input buffer sizes were used in order to transfer fixed size data from/to our decrypting function inside SGX enclaves (buffer size $\leq$ chunk size). It was done to obtain the optimum buffer size when using SGX to decrypt the video content. The size of such buffer sizes varied from 1 to 1024 Kbytes. If a buffer became incomplete by the lack of data from video chunks, it was filled out with zeroes.

7.4. Cryptographic algorithms and implementation details

We used 128-bits AES-CGM (Galois/Counter Mode) cipher for decrypting video chunks, and evaluated this process performance using both, SGX enclaves and OpenSSL. AES-GCM algorithm was selected because its implementation is already provided for SDK-SGX. Moreover, this algorithm is adopted on wireless technologies (e.g. Wi-Fi [IEEE 2011a] and WIMAX [IEEE 2011b]) and applications such as security of smart cards [Mozaffari-Kermani and Reihani-Masoleh 2012]. On the other hand, OpenSSL was chosen because it is one of the most used open source libraries, well documented and integrated to a wide range of applications [Butin et al. 2017]. Thus, we have two STB

Table 1. Video Samples List [Xiph.Org 2016]

Video Name	Resolution	Freq.	Frames	Duration
Coast Guard	360x488p	60 Hz	300	10 secs.
Foreman	360x488p	50 Hz	300	10 secs.
Ducks Take Off	1280x720p	50 Hz	500	10 secs.
Ducks Take Off	1920x1080p	50 Hz	500	10 secs.
In To Tree	1280x720p	50 Hz	500	10 secs.
In To Tree	1920x1080p	50 Hz	500	10 secs.
Old Town Cross	3840x2160p	50 Hz	500	10 secs.
Park Joy	3840x2160p	50 Hz	500	10 secs.

solutions for decrypting video chunks with AES-CGM cipher: using OpenSSL (or non SGX solution) and using SGX (running inside enclaves).

In order to implement the SGX enclave solution we use the library `libsgx_urts` offered by Intel to create and destroy enclaves. An `ECALL` to a routine inside SGX is called every time a chunk needs to be decrypted, transferring the chunk itself into the enclave with an Enclave ID (EID). The decryption was not implemented with parallel processing e.g. threads. From SGX cryptographic algorithm library (`sgx_tcrypto`), we used `sgx_rijndael128GCM_decrypt()` for chunk decryption. A key-type variable `(sgx_aes_gcm_128bit_key_t)` was set inside the enclave to hold the secret key used on the video chunks decryption. Each new video is decrypted by a different key.

A pseudo random number generator to represent the SGX function `(sgx_read_rand())` is an essential aspect of the solution. The composition of keys includes Hash-based Message Authentication Code (HMAC) and Initialization Vectors (IV). It is important to point out that a predictable random number generator can easily compromise the cryptography [Mohammad Hasanzadeh-Mofrad and Gray 2017]. The use of SGX solves this problem by generating keys in a secure space. In an actual application, the STB requests a secret key for the DRM Key Server through a secure connection defined within the HLS manifest file (* .m3u8).

At the OpenSSL solution we use the library `Crypto` offered by OpenSSL to implement STB decryption of the chunks. Equivalent to the SGX solution, here we used the High-Level Cryptographic Functions (OPENSSL-EVP) `EVP_aes_128_gcm()` mode to set the AES-GCM into OpenSSL for decrypting context. Although there is a delay involved in transferring data from/to OpenSSL libraries, there is no memory encryption mechanism since the final binary code run outside enclaves.

7.5. Performance evaluation procedure

The design of experiments was aimed to obtain the maximum information with the minimum number of experiments possible [Jain 1991]. The model for the design of experiments adopted is the full factorial. It uses all possible combinations between the levels of the factors and it is able to get the interactions between factors. The results shown are the average of the measured quantities and the confidence interval was calculated with a confidence level of 95% [Jain 1991].

8. Results and Discussions

The results were statistically equal across all sample videos presented in Table 1. Therefore we will show results from "Foreman" video.

Using SGX technology to protect the chunks may inevitably introduce additional overhead. Through our analysis, the extra overhead introduced by SGX is mainly from the aspects already reported [Gjerdrum et al. 2017] [Harnik 2017]: 1) management of enclaves (mainly creating and destroy them); 2) `ECALL`s and `OCALL`s that require the processor to perform preparations to move the execution from untrusted to trusted areas. Additionally we observe that the memory ciphering applied to the enclave memory influences the video decryption performance.

Figure 3 shows the influence of different input buffer sizes for transferring data from/to decrypting function inside SGX enclaves. We observed that input buffer sizes of 32 KB provides optimum performance in decrypting entire video chunks, even considering the penalty related to perform several enclave data transfer for small chunks in opposition to using a few large chunks. Therefore we focused on analyzing the performance related to input buffer sizes of 16, 32, and 64 KB.

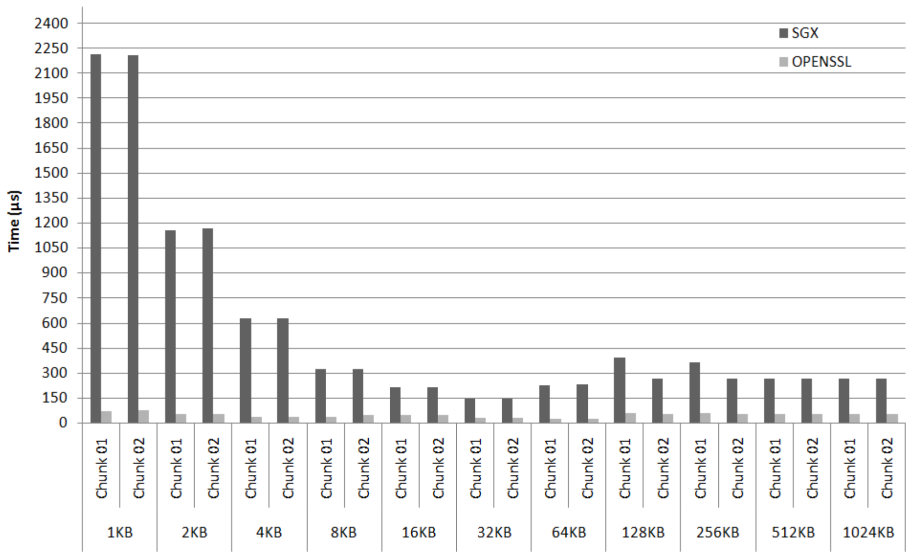

Figure 3. Average duration for SGX and OpenSSL decryption routines (video "Foreman") for input buffer sizes of 1 and 1024 Kbytes.

The encryption mean times were measured in microseconds. Chunk 01 contains video parameters and information about the subsequent chunks, and Chunk 02 contains video frames. Chunks 01 and 02 are statistically equal for results shown in Figures 4 and 5 assuming the same conditions for input buffer size. On the other hand, regarding the input buffer size, results with 32 KB performed slightly better than 16 KB in both cases (OpenSSL and SGX). Although resulting times using SGX are higher in every case, the overall improvements towards data security help providing a better solution with a reasonable decryption time that does not affect the final user experience.

Furthermore, we have also carried out an influence analysis of three important factors: factor A consists on the library (SGX and OpenSSL); factor B is the buffer size

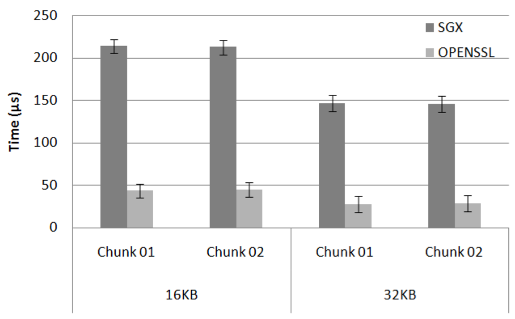

Figure 4. Average duration for SGX and OpenSSL decryption routines ("Foreman" video) for input buffer sizes of 16 and 32 Kbytes.

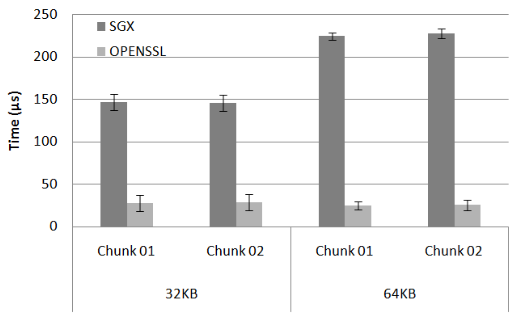

Figure 5. Average duration for SGX and OpenSSL decryption routines ("Foreman" video) for input buffer sizes of 32 and 64 Kbytes.

(16 and 32 Kbytes); and factor C represents the file sizes. For this configuration of experiments, the library influenced the most on results, representing almost 90%; the input buffer size influenced 7.59%; factor C and combinations involving factor A, B and C had irrelevant influences (Figure 6). Similar results are seen in Figure 7 when we considered buffer sizes of 32 and 64 Kbytes. As previously pointed out, although a hardware encryption technique led to a performance reduction, it is still viable for the final application of decrypting and playbacking video at the same time and potentially improves data security.

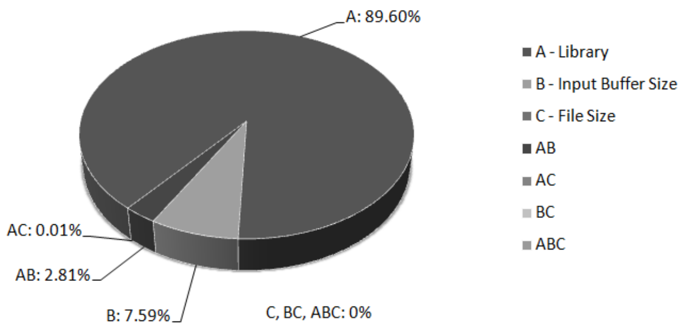

Figure 6. Influence of different factors in decrypting video chunks of size 16 and 32 Kbytes ("Foreman" video)

The full process of moving data from an untrusted area to the enclave for decryption process involves several steps. Aiming at a detailed evaluation on the impact of each step to the overall process, we have measured times according to Figure 8. Step 1 represents time spent to enter the secure area (enclave); step 2 is identified as "Others" due to the fact it represents the operations executed inside the enclave minus the decryption time itself, which is separately measured as step 3; finally, step 4 is the time spent to send data back to the untrusted area. Results shown in Figures 9 allow the analysis of times for the execution of each step.

We cannot define a relationship between decrypting duration versus input buffer size. The same is observed for duration of enclave entry and exit tasks: although the same input buffer size is used to send and receive results from enclave functions, the duration

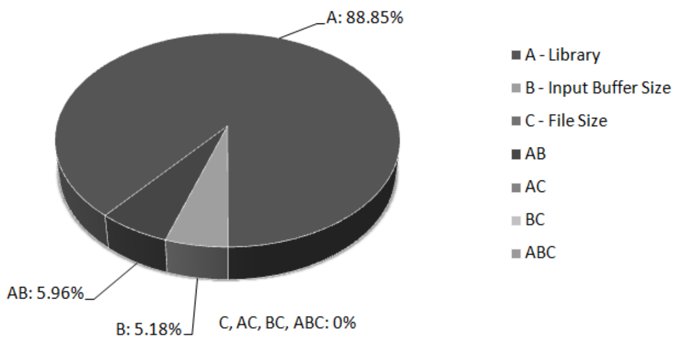

Figure 7. Influence of different factors in decrypting video chunks of input buffer sizes of 32 and 64 Kbytes ("Foreman" video)

time is different. Perhaps this should be related to the duration of memory encryption of data in enclaves. Additionally it is known that different input buffer sizes have influence on AES-GCM performance. Further analysis should be performed in order to correctly define a relationship between input buffer size and the duration of the different tasks involved in video decryption.

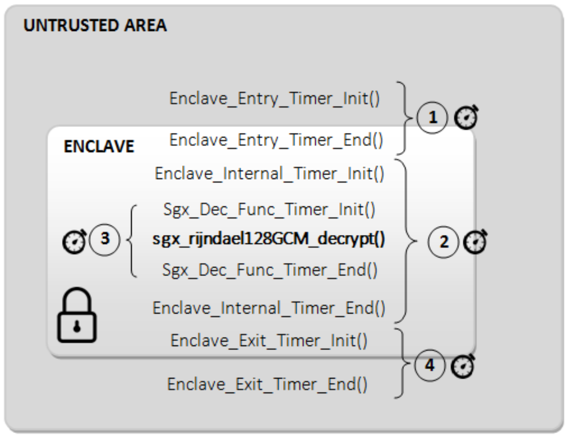
UNTRUSTED AREA
Enclave_Entry_Timer_Init()
1
ENCLAVE
Enclave_Entry_Timer_End()
Enclave_Internal_Timer_Init()
Sgx_Dec_Func_Timer_Init()
3
sgx_rijndael128GCM_decrypt()
2
Sgx_Dec_Func_Timer_End()
Enclave_Internal_Timer_End()
Enclave_Exit_Timer_Init()
4
Enclave_Exit_Timer_End()

Figure 8. Duration to accomplished tasks involved in decrypting video chunks: enclave entry (1), enclave exit (4), sgx decrypt function (3), and other enclave tasks (2).

9. Conclusions

This paper presents our preliminary results of a decrypting performance evaluation on protecting VoD content simulating a real scenario in a Trusted Environment Execution, **based on usual parameter configurations for processing video requests over a video provider network**. We demonstrated how we integrated a VoD client-side software (or STB) with Intel SGX. Our solution enables a possibility to developers to create secure applications for VoD services, leveraging the advantages of security guarantees of enclaves.

We observed the impact on video decryption performance caused by the process of transferring data from untrusted areas to enclaves (and vice-versa), along with enclave memory ciphering. We observed a minimum of $150\mu s$ in decrypting video with standard ciphers (AES-GCM in our case).

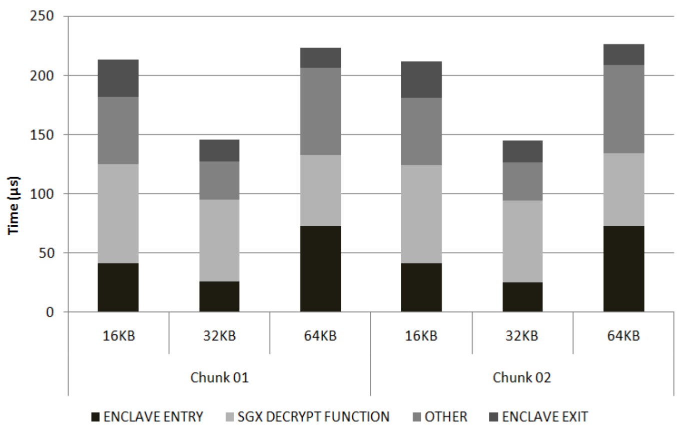

Figure 9. Performance evaluation of different tasks involved in decrypting video chunks of different input buffer sizes (16, 32, and 64 KBytes) - video "Foreman".

Future work could develop solutions using Intel SGX for: 1) Server Side: cloud-based secure applications into Content Delivery Networks to dynamically encrypt video content; 2) Client Side: secure players into embedded devices, such as: computer sticks, set-top-boxes and Over-The-Top [Mena 2017] players for tablets, smartphones and smart TVs. Other future work could be to create secure plugins to web browsers even to be an extension of W3C WebCryptoAPI [W3C 2017].

Although the tests were performed on PCs, Intel has already made available embedded devices with SGX support (their launch date was after performing most of the experiments presented in this work). These devices are still expensive compared to non-SGx similar products, but as they become more affordable, new applications for embedded systems can be economically feasible in a near future.

Acknowledgments

The work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreements 690111 (SecureCloud) and MCTIC /RNP Brazil (ACT 2549).

References

Akhyar, F., Nasution, S. M., and Purboyo, T. W. (2015). Rabbit algorithm for video on demand. *IEEE Asia Pacific Conference on Wireless and Mobile*, pages 208–213.

Anati, I., Gueron, S., Johnson, S., and Scarlata, V. (2013). Innovative technology for cpu based attestation and sealing. *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*.

Apple (2016). About http live streaming. Available at: https://developer.apple.com/library/content/referencelibrary/GettingStarted/About HTTPLiveStreaming/about/about.html.

ARM (2016). Arm trust zone: A system-wide approach to security. Available at: https://www.arm.com/products/security-on-arm/trustzone.

Butin, D., Wälde, J., and Buchmann, J. (2017). Post-quantum authentication in openssl with hash-based signatures. *Tenth International Conference on Mobile Computing and Ubiquitous Network (ICMU).*

Chakraborty, P., Dev, S., and Naganur, R. H. (2015). Dynamic http live streaming method for live feeds. *IEEE International Conference on Computational Intelligence and Communication Networks*, pages 1394–1398.

CISCO (2017). White paper: The zettabyte era: Trends and analysis. Available at: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html.

encoding.com (2016). Digital rights management - an overview. Available at: https://www.encoding.com/digital-rights-management-drm/.

FCC (2016). Fcc16-18 notice of proposed rulemaking and memorandum opinion and order.

Gjerdrum, A. T., Pettersen, R., Johansen, H. D., and Johansen, D. (2017). Performance of trusted computing in cloud infrastructures with intel sgx. In *Proc. of the 7th Intern. Conf. on Cloud Computing and Services Science. Porto, Portugal: SCITEPRESS*, pages 696–703.

Harnik, D. (2017). Impressions of intel sgx performance.

Huang, S. (2008). H264 profiles. Available at: http://blog.mediacoderhq.com/h264-profiles-and-levels/.

Hur, S., Cho, S., Kim, Y., and Sim, J. S. (2017). A power- and storage-efficient hls media server for multi-bitrate vod services. *IEEE International Conference on Big Data and Smart Computing*, 17:193–198.

IEEE (2011a). 802.11 standard. Available at: http://standards.ieee.org/getieee802/download/802.11-2007.pdf.

IEEE (2011b). 802.16e wi-max standard. Available at: http://standards.ieee.org/getieee802/download/802.16e-2005.pdf.

Intel (2016a). Intel® software guard extensions (intel® sgx). Available at: https://software.intel.com/en-us/sgx.

Intel (2016b). Overview of an intel software guard extensions enclave life cycle. Available at: https://software.intel.com/en-us/blogs/2016/12/20/overview-of-an-intel-software-guard-extensions-enclave-life-cycle.

Intel (2017). Intel software guard extensions evaluation sdk for linux os. Available at: https://01.org/intel-softwareguard-extensions.

Irawan, I. (2013). Video on demand - application of technology. Available at: http://www.almuhibbin.com/2013/01/penerapan-teknologi-video-on-demand.html.

Jain, R. (1991). *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*, page 720. Wiley professional computing, Wiley, USA, 1st edition.

Johnson, S., Zimmerman, D., and Derek, B. (2016). Intel sgx: Debug, production, pre-release what's the difference? Available at: https://software.intel.com/en-us/blogs/2016/01/07/intel-sgx-debug-production-prelease-whats-the-difference.

Knight, S. (2015). Intel to enable sgx technology on future skylake cpus. Available at: http://www.techspot.com/news/62324-intel-enable-sgx-technology-future-skylake-cpus.html.

Lal, R. and Pappachan, P. M. (2013). An architecture methodology for secure video conferencing. *2013 IEEE International Conference on Technologies for Homeland Security (HST)*, I:460–466.

Lederer, S. (2015). Optimal adaptive streaming formats mpeg-dash and hls segment length. Available at: https://bitmovin.com/mpeg-dash-hls-segment-length/.

Liebeherr, J. (1995). Multimedia networks: Issues and challenges. *Computer Magazine*, 28.

Mena, I. (2017). Verbete draft: o que é ott. Available at: https://projetodraft.com/verbete-draft-o-que-e-ott/.

Microsoft (2017). Trusted platform module technology overview. Available at: https://docs.microsoft.com/en-us/windows/device-security/tpm/trusted-platform-module-overview.

Mohammad Hasanzadeh-Mofrad, A. L. and Gray, S. L. (2017). Leveraging intel sgx to create a nondisclosure cryptographic library. Available at: https://www.groundai.com/project/leveraging-intel-sgx-to-create-a-nondisclosure-cryptographic-library/.

Mozaffari-Kermani, M. and Reihani-Masoleh, A. (2012). Efficient and high-performance parallel hardware architectures for the aes-cgm. *IEEE Transactions on Computers*, 61.

Müller, C., Lederer, S., and Timmer, C. (2012). An evaluation of dynamic adaptive streaming over http in vehicular environments. *Proceedings of the 4th Workshop on Mobile Video*, pages 37–42.

Peltoniemi, J. (1995). Video-on-demand overview. Technical report.

R. Mohan, J. S. and Li, C.-S. (1999). Adapting multimedia internet content for universal access. *IEEE Trans. Multimedia*, I:104–114.

SecureCloud (2016). Securecloud -secure big data processing in untrusted clouds (eu-br consortium). Available at: https://www.securecloudproject.eu.

Stockhammer, T. (2011). Dynamic adaptive streaming over http - standards and design principles. *Proceedings of the second annual ACM conference on Multimedia systems*, pages 133–144.

TCG (2014). Trusted platform module library. part 1: Architecture. family 2.0. revision 01.16. Technical report.

W3C (2017). Web crypto api. Available at: https://www.w3.org/TR/WebCryptoAPI/.

Xiph.Org (2016). Xiph.org video test media [derf's collection]. Available at: https://media.xiph.org/video/derf/.