White Paper

5G Broadcasting

Multi-buffer AVX-512 Accelerated Parallelization of CBCS Common Encryption Mode

Authors

Marcel Cornu (Intel),
Mark Jewett (Intel),
Sumit Mohan (Intel),
Romain Bouqueau (Motion Spell),
Tomasz Kantecki (Intel),
Gordon Kelly (Intel),
Jean Le Feuvre (Telecom Paris),
Alex Giladi (Comcast)

Abstract

The 3rd Generation Intel® Xeon® Scalable Processors with Sunny Cove core microarchitecture significantly improves AES throughput compared to previous generation processors. Throughput is improved up to 2x on legacy AES software running on Sunny Cove cores. In newly developed software, the AVX-512_VAES extension enables AES operations to be performed on 64-byte ZMM registers (up to 4 AES blocks) with a single instruction and achieves up to 3.84x improvement over the previous generation[11].

The Intel® Multi-Buffer Crypto for IPsec library's CENC CBCS implementation leverages these new features and enhancements to dramatically improve crypto performance compared to the previous generation of Intel processors. This significantly reduces the crypto overhead in multimedia packager stacks such as GPAC when doing MPEG DRM encryption and decryption by up to 15x compared to the default implementation that leverages OpenSSL[4].

Introduction

The MPEG Common Encryption (MPEG CENC, ISO/IEC 23001-7) is a DRM-independent encryption format for ISO-BMFF (mp4) files. This encryption format is widely used to protect media content delivered by adaptive streaming systems such as MPEG DASH and Apple HLS. CENC supports several encryption modes, such as CENC mode (AES-CTR) and CBCS mode (sparse AES-CBC). The latter saw rapid adoption due to its use in Apple HLS. The CBCS encryption scheme is based on AES in CBC (cipher block chaining) mode applied to every nth 16-byte block. CBC mode uses the encrypted previous 16-byte block to encrypt the current 16-byte block, thus introducing a dependency between consecutive blocks. This does not lend itself to processing multiple blocks of the same stream in parallel, while a serial implementation limits achievable performance on modern CPU architectures. A multi-buffer approach can improve the performance in CBCS mode by encrypting independent access units in parallel. This approach uses SIMD (single instruction multiple data) instructions and relies on AES-NI extensions to parallelize data processing for CBCS mode and improve performance of CENC.

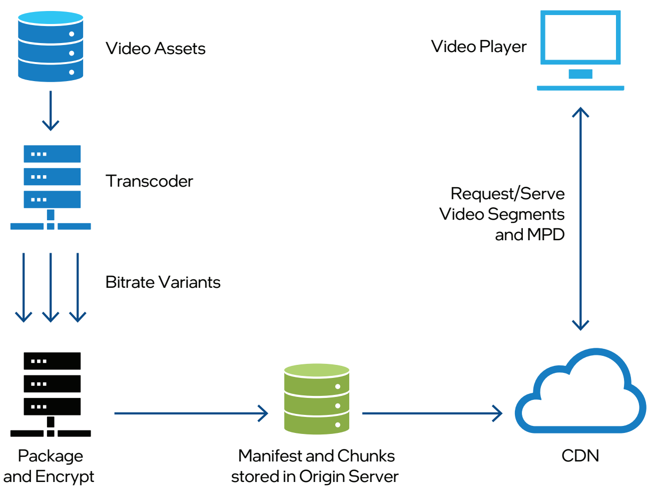
In this paper we will discuss a high-performance CBCS mode encryption implementation on Intel 3rd Generation Xeon Scalable Processors with Sunny Cove core microarchitecture bringing crypto enhancements that result in up to 3.84x throughput compared to previous generation cores. Additionally, we will discuss the CENC CBCS mode encryption implementation in the Intel Multi-Buffer Crypto for IPsec library[1], which leverages the above architectural enhancements to dramatically improve CBCS mode encryption performance in multimedia packager stacks such as GPAC[2], where we achieve up to 15x versus the default implementation (leveraging OpenSSL[4]). Since decryption is performed on individual client devices and not by a packager, this paper will focus on encryption only.

Table of Contents

Background

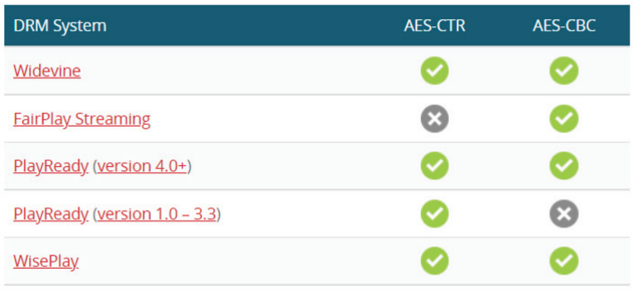
As the growth of video streaming services has continued over the last several years, so has the requirement to protect valuable content from unauthorized access. Popular streaming protocols such as HTTP Live Streaming (HLS) from Apple, Microsoft Smooth Streaming, and MPEG-DASH all use digital rights management (DRM) technologies to protect the content. DRM technologies enable content publishers to enforce their own access policies on content, such as restrictions on copying or viewing. Popular DRM technologies include FairPlay, which is Apple's DRM for HLS and works on iOS, Apple TV, and Safari on OS X. Google Widevine works with HTML5 in Google Chrome and Android Devices, and PlayReady from Microsoft. DRM uses encryption and license management to protect content and is implemented within a "Packager" in a typical video distribution workflow.

In a typical over the top (OTT) video distribution workflow, a video asset is transcoded into multiple resolutions and bitrates, called adaptive bitrate encoding (ABR), to ensure the best-possible experience for the end user by adapting to any changes in the user's network or playback conditions. After transcoding, each of the different bitrate streams are segmented into small multi-second parts, typically between 2 and 10 seconds. The individual segments are then encrypted by the "packager" using the encryption algorithm of the particular DRM technology. Different live streaming protocols support different segmentation, container, and encryption schemes. This means that several different renditions of a single asset need to be created to support playback on different devices, which can be very costly. Common Media Application Format (CMAF) and Common Encryption (CENC) emerged to make it possible to create a single rendition of an asset for distribution across numerous playback devices/platforms which use different DRM systems.

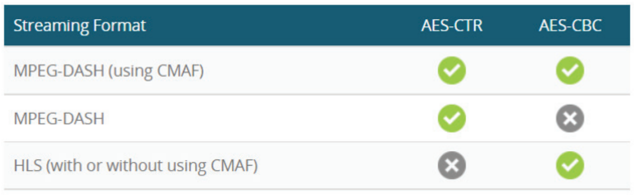

Video Assets
Transcoder
Bitrate Variants
Package and Encrypt
Manifest and Chunks stored in Origin Server
CDN
Request/Serve Video Segments and MPD
Video Player

Once the video asset has been transcoded into appropriate ABR profiles and segmented and encrypted by the packager, the content is stored on an origin server. When an end user wishes to access a particular asset, a request is made to a service which a DNS resolves to a CDN router. If the CDN has the requested asset in its cache, it is retrieved and delivered to the client device. When a customer plays the content, the player sends the license request to the content owner's platform which communicates with an authentication process. Once the validation of the user's rights to the content has been completed, the DRM platform creates the license/decryption key which is then returned to the customer's proxy and ultimately to the user's player.

CENC is an ISO 23001-7 standard that defines a common format for encryption, decryption, and key mapping methods. The primary goal of CENC was to ensure that a single file needed to be encrypted only once for distribution across numerous playback devices and platforms that use different DRM systems. The CENC encryption process is not proprietary to individual DRM systems, and video content essentially becomes DRM-neutral. CENC is supported by FairPlay, Windvine, and PlayReady when using fragmented MP4 containers and supports both AES-CTR (counter mode) and AES-CBC (cipher block chaining). AES-CTR and AES-CBC serve the same function—to protect content from unauthorized access, but fragmented support for AES-CTR and AES-CBC modes means that unless all target playback devices support a single encryption mode, two file sets are still needed today.



DRM System	AES-CTR	AES-CBC
Widevine	✓	✓
FairPlay Streaming	✗	✓
PlayReady (version 4.0+)	✓	✓
PlayReady (version 1.0 – 3.3)	✓	✗
WisePlay	✓	✓

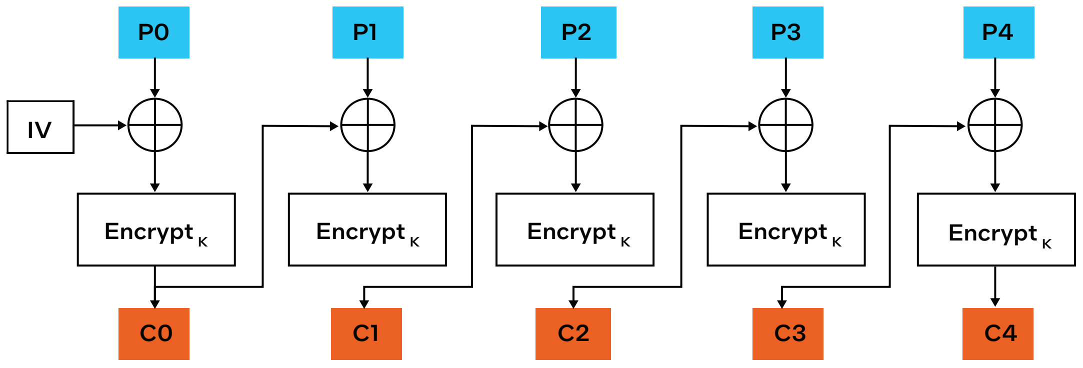


Streaming Format	AES-CTR	AES-CBC
MPEG-DASH (using CMAF)	✓	✓
MPEG-DASH	✓	✗
HLS (with or without using CMAF)	✗	✓

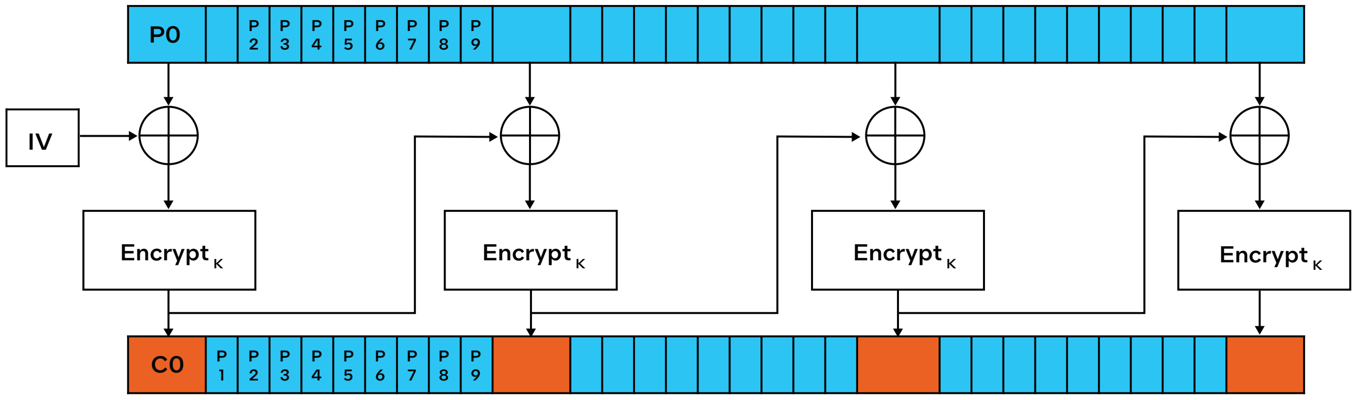
Intel Multi-Buffer Crypto for IPsec Enabling for CENC

CENC CBCS mode overview

The CENC CBCS mode encryption scheme is based on AES in CBC (cipher block chaining) mode of operation. Standard AES-CBC provides privacy by encrypting sequential 16-byte blocks of data which can later be decrypted using the same private key. The general operation of AES-CBC encryption is described as follows.



AES in CBC mode encryption



CENC CBCS mode encryption (1:9 pattern)

Encryption is performed by XORing the first plaintext block with a 16-byte initialization vector (IV) before encrypting using AES to produce a ciphertext block. Each subsequent plaintext block is XORed with the previous ciphertext block before encrypting for the remainder of the message. The dependency between plaintext and the previous ciphertext blocks means parallel processing of a single buffer is not possible.

CENC CBCS mode encryption operation is similar to AES-CBC with a configurable crypt:skip pattern where a *crypt* 16-byte block is encrypted and a number of subsequent *skip* blocks are left in the clear. A common crypt:skip pattern used is 1:9, meaning 1 block is encrypted and the following 9 blocks are left in plaintext. Decryption operates in the same manner.

CENC CBCS software implementation

CENC CBCS mode is enabled through the open-source Intel Multi-Buffer Crypto for IPsec library and GPAC multimedia framework.

Intel Multi-Buffer Crypto for IPsec Library

The Intel Multi-Buffer Crypto for IPsec library supports a wide range of confidentiality and authentication algorithms and uses various software optimization techniques to maximize CPU core utilization and crypto performance on Intel processors. SIMD instructions and AES-NI extensions are heavily used throughout the library to parallelize data processing. This paper focuses on the AVX512 implementation targeted at the Sunny Cove core. However, the library provides optimized CENC CBCS implementations for all recent Intel processors and can perform runtime feature detection to select the most optimal implementation for your hardware.

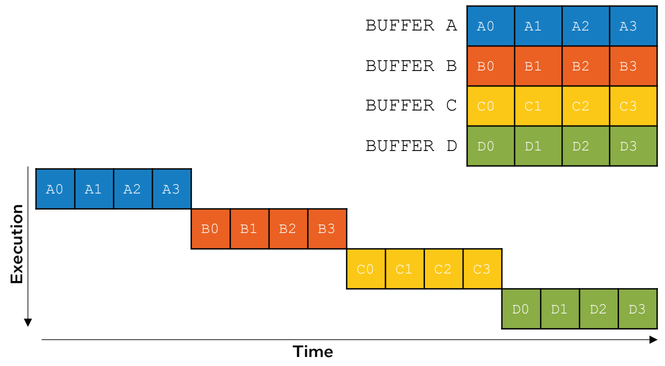
GPAC Multimedia Framework

GPAC is a popular open-source multimedia framework used in many media production chains to process and package video files. The GPAC framework supports all CENC encryption modes and currently leverages the OpenSSL[4] AES-CBC for its CBCS implementation.

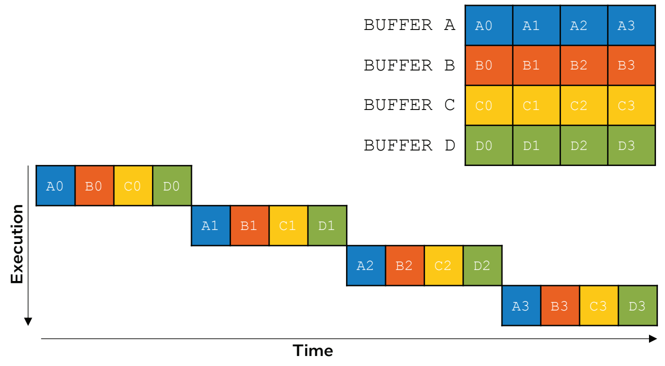
Experimental support for the IPsec Multi-Buffer CBCS mode implementation[5] was added to the GPAC framework to evaluate the multi-buffer CBCS implementation performance.

CENC CBCS Mode Encryption Implementation

The CENC CBCS encryption implementation leverages new features and enhancements available on the Sunny Cove core. Vectorized AES enhancements and the new AVX-512 VAES extension enables AES operations to be performed on full 64-byte ZMM registers (up to 4 AES blocks) with a single instruction, resulting in up to 3.84x throughput compared to previous generation cores. Refer to reference[3] for more information on 3rd Generation Intel Xeon Scalable Processor crypto enhancements.



Single-buffer processing of 4 buffers



Multi-buffer processing of 4 buffers

Single-Buffer vs Multi-Buffer Processing

The Intel Multi-Buffer Crypto for IPsec library provides optimized single-buffer and multi-buffer algorithm implementations.

The figures above show examples of how data blocks from four buffers are processed using single-buffer and multi-buffer implementations.

Single-buffer implementations process buffers sequentially and use SIMD instructions to process multiple blocks of a single buffer in parallel. Where algorithm limitations exist that prevent parallel processing of a single buffer, multi-buffer processing can be used.

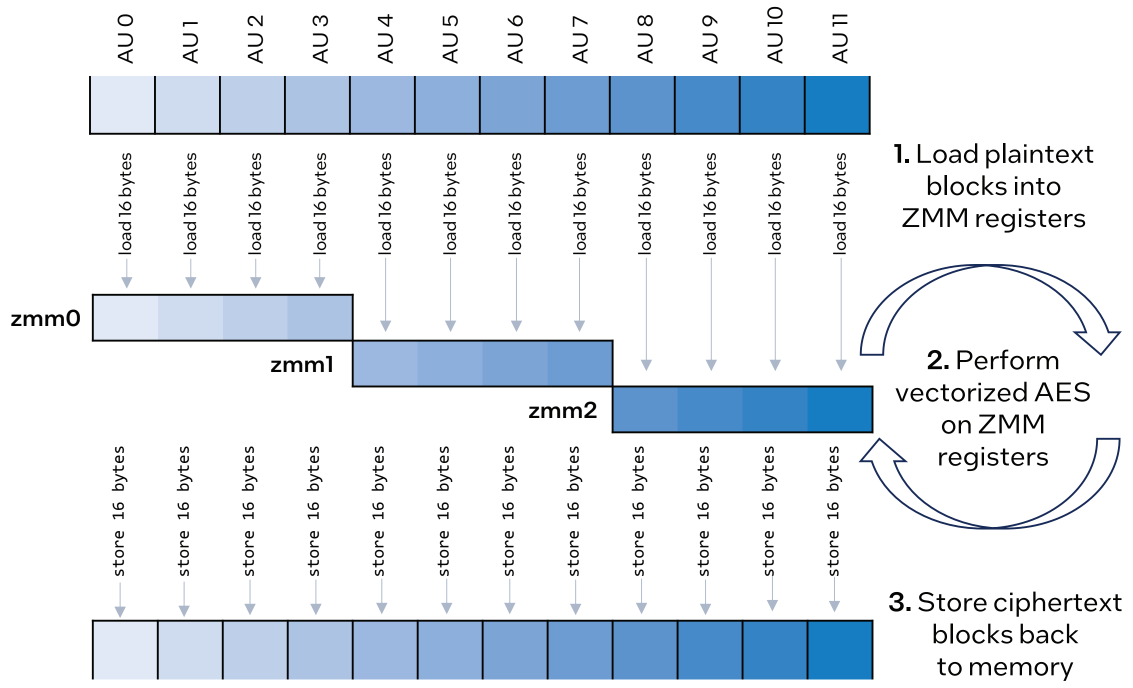
Multi-buffer implementations process single blocks from multiple buffers in parallel to maximize performance. Multi-buffer implementations add extra complexity since data must be organized correctly before processing can begin. To deal with this complexity, the library contains internal schedulers that manage the organization and processing of buffers to achieve the greatest processor utilization and throughput. In the case of CENC CBCS encryption, a multi-buffer implementation is used since it cannot be parallelized.

For more information on multi-buffer processing, please refer to reference [8].

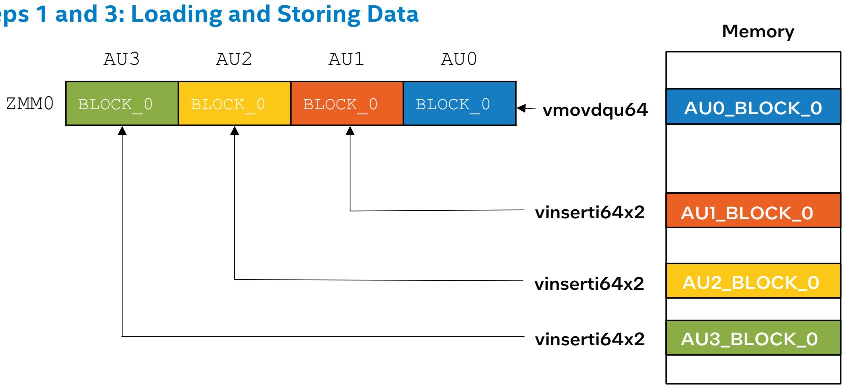
Implementation Overview

The core of the CENC CBCS implementation is performing AES encryption on multiple blocks of data in parallel. Data and AES round keys are loaded into 64-byte ZMM registers, then AES-NI vector instructions are used to process the data before storing the result.

In CENC CBCS encryption, each block is from a separate buffer and each buffer represents a single access unit (AU) of a video stream to be encrypted. On processors supporting AVX512 VAES extensions, an experimental study of critical instructions, latency, and throughput found that processing 12 AUs concurrently resulted in best performance. This is a fairly realistic scenario, as a single 60-fps video segment contains 120 AUs. For the Sunny Cove core microarchitecture, it is best to issue at least three independent AES-NI instructions. Each instruction works on four AES blocks from different AUs. Three instructions on four AES blocks = 12.


AU 0
AU 1
AU 2
AU 3
AU 4
AU 5
AU 6
AU 7
AU 8
AU 9
AU 10
AU 11
load 16 bytes
1. Load plaintext blocks into ZMM registers
zmm0
zmm1
zmm2
2. Perform vectorized AES on ZMM registers
store 16 bytes
3. Store ciphertext blocks back to memory

Steps 1 and 3: Loading and Storing Data



```
vmovdqu64   xmm0, [AU0_BLOCK_0]
vinserti64x2 zmm0, [AU1_BLOCK_0], 1
vinserti64x2 zmm0, [AU2_BLOCK_0], 2
vinserti64x2 zmm0, [AU3_BLOCK_0], 3
```

Since blocks to be encrypted are at 160-byte offsets, each block must be individually loaded into the correct index of the ZMM register. To do this, the vmovdqu64 instruction is used to load the first block into the first index, followed by three vinserti64x2 instructions to insert the remaining blocks into indexes 1, 2, and 3 of the ZMM register. Once all 12 blocks have been loaded into registers, processing can begin.

Once all rounds of AES are complete, the vmovdqu64 instruction is used again, this time to store the first ciphertext block followed by three vextracti64x2 instructions to extract the blocks in indexes 1, 2, and 3 of the ZMM register and store back to memory.

Step 2: Performing Vectorized AES on ZMM registers

```
;; Perform AES encrypt rounds on 12 buffers / AU's
;; 1 data block per buffer
;; Legend:
;; zmm0 = AU 0-3 data blocks,      zmm1 = AU 4-7 data blocks,
;; zmm2 = AU 8-11 data blocks
;; zmm3 = IV for 0-3 data blocks, zmm4 = IV for 4-7 data blocks,
;; zmm5 = IV for 8-11 data blocks,
;; rdx - pointer to AES round key table

;; Use ternary logic to do 3-way XOR with first block, IV / ciphertext and round 0 key
vpternlogq  zmm0, zmm3, [rdx + (0*12*16) + (0*16)], 0x96
vpternlogq  zmm1, zmm4, [rdx + (0*12*16) + (4*16)], 0x96
vpternlogq  zmm2, zmm5, [rdx + (0*12*16) + (8*16)], 0x96
;; AES rounds 1 to 10 using VAESENC
vaesenc     zmm0, zmm0, [rdx + (1*12*16) + (0*16)]
vaesenc     zmm1, zmm1, [rdx + (1*12*16) + (4*16)]
vaesenc     zmm2, zmm2, [rdx + (1*12*16) + (8*16)]
vaesenc     zmm0, zmm0, [rdx + (2*12*16) + (0*16)]
vaesenc     zmm1, zmm1, [rdx + (2*12*16) + (4*16)]
vaesenc     zmm2, zmm2, [rdx + (2*12*16) + (8*16)]
   ...
vaesenc     zmm0, zmm0, [rdx + (10*12*16) + (0*16)]
vaesenc     zmm1, zmm1, [rdx + (10*12*16) + (4*16)]
vaesenc     zmm2, zmm2, [rdx + (10*12*16) + (8*16)]
;; Last round using VAESENCLAST
vaesenclast zmm0, zmm0, [rdx + (11*12*16) + (0*16)]
vaesenclast zmm1, zmm1, [rdx + (11*12*16) + (4*16)]
vaesenclast zmm2, zmm2, [rdx + (11*12*16) + (8*16)]
```

The current CENC CBCS implementation uses a 128-bit key to encrypt data blocks. This operation is made up of 11 rounds and can be summarized in three steps:

1. **Initial round 0**
   First, add (bitwise XOR) the IV or ciphertext block with the next plaintext block. Next, the AES round 0 key is added to the result. The AVX512 `vpternlogq` bitwise ternary logic instruction allows these two operations to be performed with a single instruction. This reduces the number of instructions in the main processing loop and frees up the CPU port which improves overall performance of the algorithm.
2. **Rounds 1 to 10**
   This step makes up the bulk of the overall AES encryption process. The resulting data from the initial round is transformed using the vectorized AESENC instruction to perform a single round of AES encryption. Since vectorized AES on Sunny Cove core can operate 4 blocks in a single instruction, three `vaesenc` instructions are required to process all 12 blocks / AUs per round. This process is repeated 10 times using round keys 1 to 10 as the second source operand.
3. **Final Round**
   The final round uses the vectorized AESENCLAST instruction to perform the last round of AES encryption on all 12 blocks / AUs. Three `vaesenclast` instructions are issued with round 11 key as the second source operand, producing the final ciphertext blocks as the result.

GPAC Framework Modifications

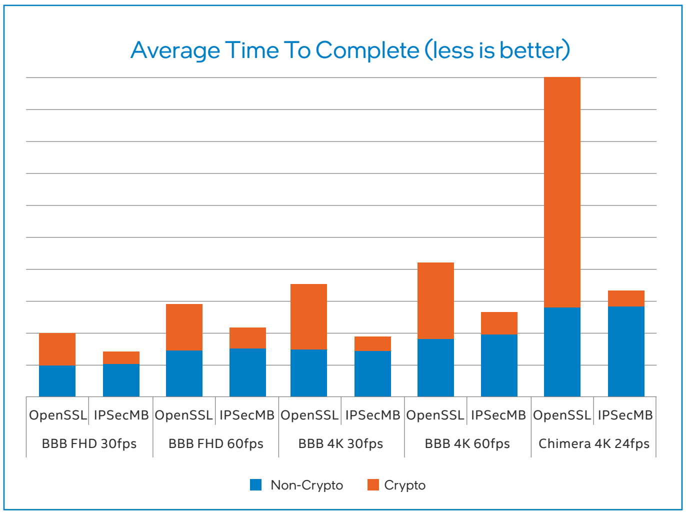
The GPAC CENC encryption filter currently supports only synchronous encryption. However, since the optimized CENC CBCS encryption operates on 12 buffers in parallel, experimental support for asynchronous processing was added. This allows data packets (buffers) to be queued until enough packets have been submitted to the IPsec Multi-Buffer library to start processing. Once processing begins, encrypted packets are returned and removed from the queue before being sent to the next destination filter for further processing.

Test Setup and Profiling

Testing was performed using the "gpac"[2] packager application to measure the time taken to complete encryption of select video files using the default (OpenSSL) implementation and IPsec Multi-Buffer. Time spent in GPAC's CENC encryption module was also recorded to determine the cost of crypto processing.

Results

Test Video File	Details
Big Buck Bunny[6]	Full HD@30fps (3Mbps), Full HD@60fps (4Mbps), 4K@30fps (7.5Mbps), 4K@60fps (8Mbps)
Chimera[7]	4K@24fps (54.4Mbps)



A significant improvement was observed when using IPsec Multi-Buffer library for CBCS encryption when compared to GPAC's default implementation leveraging OpenSSL. Crypto processing saw 55 to 90 percent improvement depending on the video bitrate. Unsurprisingly, much higher speed-ups have been achieved with higher bitrate files. The improvement when measuring the overall runtime was 25 to 65 percent, again achieving better results with higher bitrate files.

Conclusion

Combining crypto enhancements on the latest Intel® Xeon® Processors with optimized software implementations can dramatically accelerate MPEG DRM encryption. This reduces the processing resource requirements of the "packager" and reduces the total cost of ownership for an over-the-top service.

Best performance gains were observed in video streams with higher bitrates. This is due to larger buffer sizes being encrypted requiring less buffer management, and significantly more CPU time spent on encryption. With higher quality higher-rate content (such as 8K and immersive) becoming more popular, even greater efficiency could be achieved.

References

1. Intel® Multi-Buffer Crypto for IPsec library: https://github.com/intel/intel-ipsec-mb
2. GPAC Multimedia framework: https://dl.acm.org/doi/10.1145/3339825.3394929
3. 3rd Generation Intel® Xeon® Scalable Processor crypto enhancements: https://newsroom.intel.com/articles/crypto-acceleration-enabling-path-future-computing/#gs.n34xtw
4. OpenSSL: https://www.openssl.org
5. GPAC - IPsec Multi-Buffer asynchronous prototype: https://github.com/mdcornu/gpac/tree/async_cryp_ipsecmb
6. Big Buck Bunny video files: http://bbb3d.renderfarming.net/download.html
7. Chimera video file: http://download.opencontent.netflix.com
8. Processing Multiple Buffers in Parallel to Increase Performance on Intel Architecture Processors: https://github.com/intel/intel-ipsec-mb/wiki/doc/communications-ia-multi-buffer-paper.pdf
9. CENC CBCS example code: https://github.com/intel/intel-ipsec-mb/wiki/MPEG-CENC-in-CBCS-mode
10. For documentation on vector AES-NI instructions, see volume 2, chapter 3 of the Intel Software Developers Manual: https://software.intel.com/content/www/us/en/develop/articles/intel-sdm.html
11. 3rd Gen Intel® Xeon® Scalable Platform Press Presentation (Appendix 20): https://newsroom.intel.com/wp-content/uploads/sites/11/2021/04/3rd-Gen-Intel-Xeon-Scalable-Platform-Press-Presentation-281884.pdf

Test Environment and Configuration

Test by Intel as of December 10th 2021	
Platform	Inspur NF5180M6
CPU	2S Intel(R) Xeon(R) Gold 6348 (2.60GHz)
Microcode	0xd0002a0
Hyperthreading	Disabled
Turbo	Disabled
Operating System	Ubuntu 20.04.03 LTS (5.4.0-91-generic)
Codec	H.264

Solution provided by:

intel®