**INTERNATIONAL ORGANIZATION FOR STANDARDIZATION**

**ORGANISATION INTERNATIONALE DE NORMALISATION**

**ISO/IEC JTC 1/SC 29/WG 03 MPEG SYSTEMS**

**ISO/IEC JTC 1/SC 29/WG 03 N1560**

**July 2025 – Daejeon, KR**

|  |  |
| --- | --- |
| **Title** | **Exploration on De-CENC related issue** |
| **Source** | **WG 03, MPEG Systems** |
| **Status** | **Approved** |
| **Serial Number** | **25307** |

# Introduction

In September 2024, phrack.org published an article titled “[MPEG-CENC: Defective by Specification](https://phrack.org/issues/71/6.html#article)”. The article outlines an attack, called De-CENC, that compromises the Secure Video Path to enable the capture of decrypted, compressed content without breaching DRM protections. In response to this attack, MPEG is evaluating potential mitigations.

This document provides a detailed description of the attack, guiding principles, and potential solutions, which may form the basis of a future amendment to the Common Encryption (CENC) specification.

# Attack Description

The core principle exploited by the De-CENC attack is to inject new bitstream elements around and into an encrypted “original” bitstream, creating a new “meta bitstream” that wraps the original bitstream. This wrapping prevents or bypasses the decoding of the original bitstream, while still allowing decryption by the Content Decryption Module (CDM). The result is a side channel which enables inferring the binary data of the decrypted (wrapped) bitstream when decoding the meta bitstream.

One way to achieve this is to create a meta bitstream that displays the binary data of the wrapped bitstream as a visual pattern on screen, which can then be digitized. This is demonstrated by De-CENC using I\_PCM macro blocks in the meta bitstream, embedding the wrapped bitstream as payload data. The I\_PCM macro blocks are specifically crafted to render their payload bytes as grayscale dots when decoded. Other attack variants may construct a meta bitstream which leaks information more subtly, but also at a lower rate than what is achievable with codec constructs like I\_PCM, which can print information verbatim on screen.

The attack can either target directly the decrypted original bitstream or aim to extract the AES CTR mode keystream of ‘cenc’ content by creating a decryption oracle, and decrypt the original content in a second step. This is possible because in AES CTR mode the ciphertext gets decrypted by XORing it with the keystream, which means the keystream can be derived by XORing the decryption output with the ciphertext.

When aiming at extraction of the AES CTR mode keystream, the original bitstream can be replaced with random values. This approach avoids that valid bitstream elements in the wrapped bitstream interfere with the injected bitstream, and simplifies working around luma clamping in various video formats by generating two variants of the random input (with toggled MSB) for the same keystream block, ensuring one output stays within the clamped luma range.

In contrast, AES CBC mode doesn't have a keystream, so the original encrypted bitstream of ‘cbcs’ encrypted content needs to be directly attacked. To work around the aforementioned decoding challenges, another way of altering the decryption output in a controlled manner is required. In AES CBC mode the IV (or the previous block’s cipher text) is used to XOR the output after decryption, which can be leveraged to modify the decrypted bitstream by using a random IV (thereby randomizing the decryption output). The process can be reversed after rendering the decrypted data on screen, ultimately applying the original IV to arrive at the correct decrypted output.

This means that while the primary principle of the attack doesn’t interfere with decryption itself, it requires modification of IVs and/or encrypted payloads to work around decoding and rendering issues of the injected meta bitstream, when leaking the decrypted wrapped bitstream.

One specificity of the De-CENC attack is to leverage CENC subsample encryption to inject the new bitstream into expanded clear ranges. But other attack vectors are available as well, e.g. sample groups with unencrypted samples, which are used to implement the “clear lead” concept and certain ad-insertion use cases.

Applicability to:

* Any codec
* Any cipher mode: CTR, CBC
* Any DRM system
* Any protection scheme defined in MPEG-CENC

Focus on some parts of the attack and some extensions:

* use of subsample encryption
* use of I\_PCM block
* use of clear lead

# Guiding Principles

The design of any solution should:

* Protect any encoded video stream, regardless of coding tools, including constructs like I\_PCM.
* Consider both backward-compatible and non-backwards-compatible approaches.
* Optimize for integration with the content pipeline.
* Minimize impact on end-user device performance.
* Consider legitimate use cases for the manipulation of encrypted data (e.g. HEVC tile composition).
* Leverage opt-in profiles to prohibit risky coding tools such as PCM constructs.
* Consider integrity boundaries for efficient hardware support.

If the integrity enhancements are developed, it is essential to preserve compatibility across a wide range of existing devices and content. Many deployed systems, particularly those with hardware limitations, may not receive timely updates. Furthermore, significant volumes of legacy content should remain accessible and cannot feasibly be reprocessed, re-encrypted, or duplicated. Therefore, any proposed solution shall accommodate both legacy and updated environments, enabling a seamless transition.

The solution should enable the production of new content that incorporates integrity protection while retaining compatibility with older playback systems. Updated players shall be permissive with legacy content to ensure backward compatibility, while enforcing enhanced integrity constraints for new content.

# Considerations for a Solution

This section outlines solution elements with associated trade-offs. They include use of existing ISOBMFF structures as well as potential new structure definitions, discuss compatible approaches as well as non-backwards compatible approaches.

## Authenticated Encryption Applied to CENC

### Approach 1

Replace the current encryption algorithms (AES-CTR, AES-CTR) with an Authenticated Encryption algorithm. The unencrypted data can be hashed using SHA1 or SHA256 and then provided to the algorithm as Additional Associated Data (AAD). Algorithms: AES-GCM, AES-GCM-SIV, ChaCha20-Poly1305.

1. Like AES-CTR, AES-GCM is not secure if the nonce (IV) is reused, where AES-GCM-SIV does not have the issue. Thus the nonce (IV) needs to be unique for AES-GCM. The AES-GCM runs much faster than AES-GCM-SIV.
2. ChaCha20-Poly1305 is faster than AES-GCM in pure software implementations, but AES-GCM can take advantage of hardware acceleration (AES specific instruction), which is widely available.
3. Winner: AES-GCM, but need to make sure that the nonce (IV) is unique.

### Approach 2

Keep the existing encryption algorithms (AES-CTR for cenc, AES-CBC for cbcs), and perform authentication after encryption. Algorithms: HMAC-SHA1, HMAC-SHA256, Poly1305.

1. HMAC-SHA1 has better performance than HMAC-SHA256, but HMAC-SHA256 has better collision resistance.
2. Poly1305 is fastest, but it is not as widely available as HMAC. Note that Poly1305 requires a unique nonce for every message.

Compared to approach 1, approach 2 is backward compatible and is more flexible.

1. Allows the encrypted content to be processed and the authentication tag to be re-generated without the need to decrypt the content first.
2. The authentication key can be different to the encryption key, and can be shared with different entities.
3. Compatible with the existing encryption schemes and the existing implementations, and also easier to implement on top of the existing implementations.
4. Allows license providers to choose whether to enforce the authentication on the server side on a per content basis.

Rough performance comparison measured on a Google Cloud VM, assuming average sample size of 26KB.

| **Algorithms** | **MB/s** |
| --- | --- |
| *AES-CTR* | *752* |
| *AES-CBC* | *924* |
| AES-GCM | 414 |
| AES-GCM-SIV | 96 |
| ChaCha20-Poly1305 | 263 |
| AES-CBC + HMAC-SHA256 | 581 |
| AES-CTR + HMAC-SHA256 | 508 |
| AES-CBC + Poly1305 | 647 |
| AES-CTR + Poly1305 | 558 |
| *AES-CBC 1:9* | *9240* |
| AES-GCM 1:9 sha256 | 1227 |
| AES-GCM 1:9 poly1305 | 1521 |
| AES-CBC 1:9 + HMAC-SHA256 | 1342 |
| AES-CBC 1:9 + Poly1305 | 1754 |

### Approach 3

#### Cryptographic Scheme

The following cryptographic configuration was recommended by m73651:

* **Encryption algorithm**: AES in CBCS mode (as per current CENC specification)
  + AES with 128-bit keys should be mandated:
    - Widely supported in legacy software and hardware.
    - Sufficient for DRM use cases even considering potential quantum threats.
  + AES with 256-bit keys may be optionally supported but is not a current requirement. A corresponding amendment is under development.
* **Integrity algorithm**: Galois Message Authentication Code (**GMAC**) , also known as AES-GCM, only used with authenticated data.

The presence of integrity protection shall be explicitly signalled in the file to ensure correct decoding behavior. The recommended authentication approach is to define a new axis of sample auxiliary information, analogous to existing subsample range signalling, which conveys hash digests computed over specified spans of sample data.

This configuration is **favourable for hardware implementations** and aligns well with software that benefits from AES-GCM instructions. GMAC provides efficient, low-latency integrity verification while maintaining compatibility with AES-based encryption workflows already in use.

As we propose the use of GMAC for integrity protection, it is necessary to define the associated key usage model. Integrity protection shall use a distinct cryptographic key, separate from the AES-CBCS content encryption key. This key shall be conveyed via a secure channel, such as the license response, alongside the encryption key. This approach provides clear cryptographic separation between encryption and authentication.

#### Integrity Scope

The entire sample shall be authenticated prior to encryption, or equivalently, after decryption. This ensures that any modification to the data presented to the decoder, including slice structure, and coded payload, is detectable. The objective is to prevent attackers from manipulating decoder inputs to influence behavior or extract sensitive information.

For NAL-based video formats, any NAL unit that contains a protected region shall be covered by a GMAC digest. The authenticated region shall consist of the NAL unit payload, **excluding** both the NAL unit header and the start code or length field. This includes both clear and encrypted byte ranges and ensures that the meaningful decoder input is verified without interfering with stream parsing or packetization.

In order to support enforcement in hardware, the use of integrity protection shall be signaled explicitly. This may be indicated through a new file brand that identifies integrity-protected samples. However, because file brands are not inherently secure, this signaling shall also be delivered through a trusted mechanism such as a license message, a key response, or another secure channel.

This model ensures that only fully authenticated sample data is accepted by the decoder, providing a robust defense against attacks that rely on tampering with decoder-visible structures.

#### Signaling in ISOBMFF

It is proposed that integrity protection be signaled using the existing 'saiz' and 'saio' mechanisms defined in the ISOBMFF. These structures allow the association of authentication data, such as GMAC tags, with media samples, without altering the media payload. This approach supports efficient signaling and remains compatible with subsample encryption workflows.

For NAL-based video codecs, the integrity should be applied at the level of individual NAL units that contain protected regions. The authenticated region would exclude the NAL unit header and the start code or length field used for framing. Metadata such as SEI messages is not covered, as it is typically ignored by hardware and does not influence decoder state. Authenticated regions are defined by decoder input not by encryption boundaries. This allows us to flexibly apply integrity protection even over clear (non-encrypted) data.

In the case of in-band tracks, such as those using the 'hev1' sample entry, parameter sets (e.g., VPS, SPS, PPS) are embedded within the media samples and are not typically parsed or validated by hardware decoders. As such, we do not consider it practical or beneficial to authenticate these elements. The associated attack surface is limited, and the implementation cost of protecting parameter sets is not justified by the security gains. Instead, integrity protection should focus on slice data, which directly influences decoder behavior and is actively processed by hardware, making it a more meaningful and effective target for authentication.

## Protecting Initialization Vectors

As mentioned in the attack section, the IV can be manipulated to craft attack controlled bitstreams if not protected, so the IV must be either authenticated or included in the derivation of the authentication key, e.g. if Poly1305 is chosen as the authentication algorithm.

How can the IV be authenticated:

1. AES-GCM: aad = SHA256(IV + …), output..auth\_tag = AES-GCM(...)
2. HMAC: auth\_tag = HMAC(IV + …)
3. Poly1305: poly1305\_key = HMAC(IV), auth\_tag = Poly1305(...)

## Protecting Sample Auxiliary Information or Sample Data

**Method 1**: Protecting Sample Auxiliary Information alone (IV + number of bytes in subsamples) is not sufficient

1. There is a high possibility that attackers can craft I\_PCM bitstreams matching one of the authenticated ranges.

**Method 2**: How about protecting Sample Auxiliary Information (IV + number of bytes in subsamples) and all the encrypted bytes?

1. It is still possible to craft I\_PCM bitstreams by manipulating the ClearData alone with some range values.
2. There are other primitives that can be used to leak data.
3. Need further investigations to understand the risk.

**Method 3**: Protecting all the data including the unencrypted data in ClearData and SkipByteBlock

1. Most secure but with the worst performance regression, >30% as shown above.
2. There are two variants: (1) Generate one authentication tag per sample; (2) Generate one authentication tag per subsample + one optional authentication tag for the Sample Auxiliary Information.
   1. The first option is more secure but the second option is more flexible.

## Current support in HW/SW

HMAC-SHA256 is available in all the current HW / SW.

AES-GCM should be available in most HW/SW - need confirmation.

ChaCha20-Poly1305 was added to MbedTLS, one of the most widely used crypto libraries for Trusted Applications, in 2018. It should be available to use in most TEE OS now - need confirmation.

### Integrity Boundaries for Efficient Hardware Support

An important requirement is that **the size of data covered by each integrity tag shall be bounded**, to support fast, deterministic validation by hardware:

* The **initial recommendation** is that the specification shall support signalling a maximum **integrity block size of 16 KB**. Allowing this limitation will provide better security: given the size of the internal private buffer, video decrypting and decoding hardware will start decrypting and decoding a frame if and only if the block integrity is correct, and that without any software interaction.
* From a performance standpoint: Decoding can start before the entire frame is authenticated, reducing the risk of pipeline bubbles.
* If a frame exceeds the maximum size, it shall be split and covered by multiple integrity tags to ensure full protection.

Failure to define such a limit exposes a potential attack vector, as software-based verification becomes a point of weakness compared to hardware-based mechanisms.

## Box Descriptions

### Adding Authentication Tag to CencSampleAuxiliaryDataFormat

Add the authentication tag to CencSampleAuxiliaryDataFormat and the per sample section of SampleEncryptionBox:

aligned(8) class CencSampleAuxiliaryDataFormat {

if (aux\_info\_type\_parameter == 0) {

unsigned int(Per\_Sample\_IV\_Size\*8) InitializationVector;

if (sample\_info\_size > Per\_Sample\_IV\_Size ) {

unsigned int(16) subsample\_count;

{

unsigned int(16) BytesOfClearData;

unsigned int(32) BytesOfProtectedData;

} [subsample\_count ]

unsigned int(256) auth\_tag;

}

}

...

}

Issues:

1. Not backward compatible. Existing implementations, which check the version or flag field, will fail to parse the new format.
2. Other existing implementations will assume the old format and return with incorrect data (e.g. Chrome), which will result in decoding failures.

### Appending Authentication Tag to Encrypted Samples in mdat

Add the authentication tag to the end of encrypted sample in mdat

Pros:

1. No additional mp4 boxes or fields that need to be parsed.
2. No Decryptor API changes needed to pass in the authentication tags, although the Decryptor still needs a way to know whether the authentication tags are present in the bitstream, but it can be implicit.

Issues:

1. Not backward compatible. Existing Decryptor implementations that check Sample Size == SUM(Subsample Sizes) will fail.

### Defining a New SampleAuthBox for Authentication Tags

Add a new box to carry authentication tags:

aligned(8) class SampleAuthBox extends FullBox('saut', version, flags) {

unsigned int(32) sample\_count;

{

unsigned int(256) auth\_tag;

}[ sample\_count ]

}

Pros: backward compatible

1. Allows contents to be re-packaged without breaking existing clients, if the server chooses not to enforce the authentication on these clients.
2. The content packaging and the server enforcements are two separate processes.
3. Content providers can choose to enforce the authentication only when needed on the server side.

### Appending HMAC to SampleEncryptionBox

Add HMAC to the end of SampleEncryptionBox

aligned(8) class SampleEncryptionBox extends FullBox('senc', version, flags) {

unsigned int(32) sample\_count;

// IV and subsample info.

{

…

}[ sample\_count ]

// HMACs.

{

unsigned int(256) auth\_tag;

}[ sample\_count ]

}

Pros: backward compatible. See above.

## Prohibition of PCM-like Constructs

Certain encoding constructs, referred to here as “PCM-like” constructs, permit raw or minimally processed data to be passed through directly from the encrypted bitstream to the decoded visual output. While these constructs are rarely, if ever, used in production content, they are known to enable attacks such as De-CENC, where decrypted bytes may be leaked through visual channels.

To address this, we propose the introduction of a **new CENC profile** that explicitly prohibits PCM-like constructs in encrypted video streams. This profile will be identified by a dedicated brand (e.g. 'npcm') and will include a testable list of tools prohibited in samples of an encrypted video track. The list of tools covered by this profile should be explicitly defined, enumerating all constructs that allow unverified byte values to be rendered directly as pixel data. This initial list will include I\_PCM macroblocks in AVC, PCM samples in HEVC and VVC. The list should be reviewed by MPEG in consultation with JVET, with the goal of identifying any additional tools that warrant inclusion. Once a brand is defined, its associated list of prohibited tools is fixed and does not change. If additional tools are identified in the future, a new brand or version may be defined to cover an expanded set of constructs.

Files conforming to this profile will indicate compliance by including the brand in the file type box. Content producers and service providers can check their content, and if no prohibited constructs are present, set the brand to signal conformance.

It is important to note that brand signaling within the file type box alone **does not provide cryptographic assurance** since an attacker can easily modify this signaling. Implementations should ensure that the profile signaling occurs within a **trusted domain** or is **otherwise protected against tampering**. The exact method of secure signaling may vary by deployment and is left to implementation, but the specification recommends that brands be used only when authenticity can be assured.

Other standards development organizations and derived specifications are encouraged to adopt a similar approach, defining clear brands and prohibited tool lists for their own codecs or encryption frameworks to ensure testability and security alignment.

### Possible specification text

**CENC Profile for Exclusion of PCM-like Constructs and Signaling**

Certain encoding constructs, referred to as PCM-like constructs, allow raw or minimally processed byte values to be passed through directly from the encrypted bitstream to the decoded visual output, bypassing prediction, transformation, or entropy decoding. The presence of such constructs in encrypted streams presents a serious security risk. In particular, these tools may be used to leak decrypted data through visual channels.

To address this risk, a dedicated profile for CENC is defined that excludes PCM-like constructs in encrypted video samples. This profile is optional and is indicated by the presence of the brand 'npcm' in the compatible\_brands field of the FileTypeBox.

If a file includes the 'npcm' brand, then all encrypted video samples shall not contain any coding tool or construct listed as prohibited for the brand. The prohibited tools are:

* I\_PCM macroblocks as defined in ISO/IEC 14496-10 (AVC)
* PCM samples as defined in ISO/IEC 23008-2 (HEVC)
* ...

Files shall only declare the 'npcm' brand if no encrypted video samples employ any of the prohibited constructs.

The presence of the 'npcm' brand in the FileTypeBox alone does not provide cryptographic assurance of compliance. Implementations shall ensure that the signaling of this profile is delivered and validated in a trusted environment, using out-of-band mechanisms where necessary. The specific signaling method is implementation-specific; however, in untrusted environments, secure, integrity-protected signaling is strongly recommended. In trusted domains, implementers may choose to rely solely on the file brand itself.

Derived specifications are encouraged to define similar opt-in brands and profiles for their codecs or encryption frameworks, including explicit lists of prohibited tools to ensure testability and alignment with this approach.