Text

Description automatically generatedISO/IEC JTC 1/SC 29/WG 03 N01627

**ISO/IEC JTC 1/SC 29/WG 03  
MPEG Systems   
Convenorship: KATS (Korea, Republic of)**

**Document type:** Output Document

**Title:** Exploration on De-CENC

**Status:** Approved

**Date of document:** 2025-10-17

**Source:** ISO/IEC JTC 1/SC 29/WG 03

**No. of pages:** 22 (with cover page)

**Email of Convenor:** young.L @ samsung . com

**Committee URL:** <https://isotc.iso.org/livelink/livelink/open/jtc1sc29wg3>

**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 N01627**

**October 2025, Geneva, CH**

|  |  |
| --- | --- |
| **Title** | **Exploration on De-CENC** |
| **Source** | **WG 03, MPEG Systems** |
| **Status** | **Approved** |
| **Serial Number** | **25601** |

# 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 tricks the decoder and allows the output of compressed content instead of uncompressed content without breaching DRM protections. In response to this attack, MPEG is evaluating potential mitigations.

This document continues the exploration started with “WG03N1560\_25307\_DeCENC\_Exploration” from July 2025. It provides a rationale presentation of the problem space, high-level presentations of the streaming ecosystem, considerations as we explore the solution space, a discussion of a few of the possible mitigations, and recommendations for the next steps.

# Problem space

## Attack 101

By manipulating the packaged content, particularly the codec construct, an attacker can trick the decoder into not decoding the content, and instead of sending uncompressed content on the (HDMI) output, send the compressed content.

If at a high level we define the following

* The input, an encrypted compressed stream
* The output, a decrypted uncompressed stream
* The CDM, in charge of decrypting the content
* The decoder, in charge of uncompressing the content

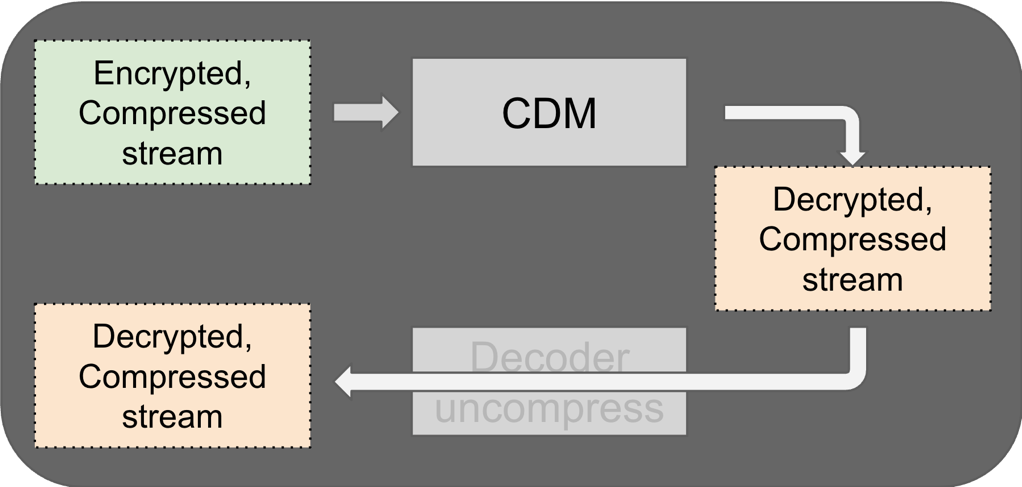
The figure below is a very high-level diagram of a “normal playback” (without DeCENC. What is expected as an output is decrypted, uncompressed content.

A diagram of a computer process

AI-generated content may be incorrect.

**Figure 1: normal playback**

When an attacker leverages the DeCENC attack, it can trick the decoder into output decrypted compressed content, as seen in the diagram below.



**Figure 2: DeCENC playback**

The diagram could be refined with more details, as in practice, the decoder is the one tricked into making the decision not to uncompress the parts of the content that should be. But for the purpose of this Attack 101 (simplification), this is a balanced level of detail for a high-level understanding of what the attack does.

## Attack novelty

What is not new: DeCENC does NOT introduce a novel way to break output protections (e.g., HDCP) and does NOT allow exfiltration of content that would have otherwise remained confidential (e.g., DRM-encrypted). In other words, it does NOT present a novel way to acquire and decrypt protected content on unauthorized clients or accounts.

What is new: By tricking the decoder not to decode, DeCENC allows the output compressed content instead of uncompressed content at the end of the video playback pipeline.

## Adversaries Pros and Cons

**DeCENC is a piracy optimization**: from a streaming device, it allows extracting compressed content then redistributing immediately (DeCENC enables “WebDL”) *versus* extracting uncompressed content then re-encoding and then redistributing (“WebRip”).

Pros: for redistributing content, it presents the following advantages: no need for re-encoding the content, or of having access to the original encodes (supposedly more efficient in size-to-quality ratio).

Cons: this is a complex attack to pull, and it would be difficult to productize on a system for repeatable and stable content extraction. Also, it requires modifying the packaged content, which in turn requires having control of the streaming application and/or the system it executes on.

## Sizing the problem

## The involved parties in the MPEG group for DeCENC mitigation have no data evidence supporting that adversaries are actively leveraging or trying to leverage DeCENC for piracy. It is complex piracy optimization to productize and does not push the boundaries of piracy or render new piracy models that would otherwise be impossible. **The DeCENC attack does not present an additive material risk or cost for content owners today**.

## Mindset & rationalization

DeCENC creates a precedent. It points to a weakness and while this attack may not be easily productized by adversaries, it is an inspiration and could eventually become a bigger problem in the future. **DeCENC introduces a new class of attacks** that aims to exploit the complexity of the encoding to trick the playback device components, here the decoder.

While not critical yet, **it is time to explore mitigation options before it is too late**. We want to explore multiple solutions that would yield value on different timelines, with **the highest priority being the right long-term solution to address the entire class of attacks**.

# Considerations on the streaming ecosystem

## Domains

There are two critical domains that need to be accounted in the streaming ecosystem. On one hand, we have the domain of encoding technologies where the goal is to innovate and create more efficient codecs, which try to the quality of the rendered media on clients while optimizing the bandwidth needed to obtain that quality. On the other hand, we also have the domain of content protection that evolves to protect the assets of content owners and their ability to monetize. Both domains are complex, and they come with different challenges and evolution timelines.

As we are considering an aspect of content protection to mitigate DeCENC, there might be relations between the two domains, but **we should pay attention to the impact on the encoding technologies and their ability to keep innovating**.

## Types of devices & ecosystems

There are many types of streaming devices: Smart TVs, Set-Top Boxes, Browsers, Phones, etc.

Those different types are powered by various systems and security architectures; they depend on different partnership stacks and related market dynamics.

In addition, the consumer relationship with those devices is different, and their overall lifecycle and operational services differ as well. A managed Set-Top Box, for example, has a different lifecycle and updateability than a TV or a tablet.

Overall, **there is no one rule when it comes to the evolution of design and the updateability of systems**; there are many perspectives that need to be accounted for when we approach a more generic problem like DeCENC.

## REE and TEE

The **R**ich **E**xecution **E**nvironment (**REE**) refers to the **system that the main/application CPU is executing from**. In general, it comprises a full operating system and often an application store. It is inherently open by nature. In general, people are in full control of the REE when they have ‘rooted’ their device, but there is granular access to REE execution, from closed systems without application store, to managed-only application stores, to the ability to side-load applications or even update/reconfigure the operating system itself.

The **T**rusted **E**xecution **E**nvironment (**TEE**) is a **secure subsystem that is not accessible by the main/application CPU**. For the most part, the TEE is fully managed by the SoC vendor and/or the OEM. No mainstream application can be loaded by the user in the TEE. The execution of the TEE and assets is fully managed. The TEE was introduced for balancing the ever Richer nature of the REE and provides a trusted environment for executing sensitive codes that safeguards the most valuable assets even in the case of a compromised system (REE).

A screenshot of a computer

AI-generated content may be incorrect.

**Figure 3: REE & TEE domains**

The mental model for this exercise is: **to exploit DeCENC the adversary needs to have access to the system on the device**. As such, REE-domain is considered fully compromised, while the TEE-domain remains trustworthy.

## SVP

The **S**ecure **V**ideo **P**ath (**SVP**, also known as Secure Video Pipeline, or Secure Media Pipeline when it also covers Audio) **provides guarantees that the content remains inaccessible from the main/application CPU during playback**. The SVP is traditionally configured by a trusted entity in the device, the TEE. It includes (non-exhaustive list) the various hardware blocks required for playback, from DRM logic to hardware decryptor, video decoder, and output. It also includes the buffer architecture and permission logic to allow the implementation of the various playback functions while remaining restricted and secure.

A diagram of a video decode

AI-generated content may be incorrect.

**Figure 4: Video pipeline with DRM, high-level components**

Some devices may have a notion of an SVP even when orchestrated from the REE. We could debate this, but in the context of this exercise around DeCENC, we will consider the SVP to only be a valid construct in the existence of a TEE.

Incorporating the above components in a broader “system” view with the TEE results in the following more complete diagram. It may not be entirely accurate on some devices, but it should be a valid high-level architecture on clients with a TEE overall.

A diagram of a computer process

AI-generated content may be incorrect.

**Figure 5: TEE/SVP System view**

## DRMs and security levels

**D**igital **R**ights **M**anagement systems (**DRM**s) provide **a scheme for content protection**. On the device, it integrates a **C**ontent **D**ecryption **M**odule (**CDM**) that is in charge of establishing a secure communication channel between the device and the backend DRM server, identifying the device and the group it belongs to, performing license acquisition, content decryption, and configuring the output protections as required inside the license.

Software (**SW**) **DRM** refers to **the DRM implementation of the CDM integrated on the application (main) CPU**, or, said otherwise, in the REE. Under such circumstances, the CDM protects the assets in the same context that is under the control of the local adversary.

e.g., Widevine L3, Playready SL2000

Hardware (**HW**) **DRM** refers to **the DRM implementation of the CDM that has a layer terminated/integrated inside a secure subsystem, the TEE**. This layer in the TEE is in charge of managing the most sensitive assets and performing the related security operations. For example, the DRM root-of-trust, session keys, and content keys are managed in the TEE, and the content is protected by the Secure Video Path, as configured and enforced by the TEE.

e.g., Widevine L1, Playready SL3000

## SW vs FW vs HW updates timelines

The difference between software and firmware can be subtle. Let’s call software anything meant to run on the application CPU (e.g., operating system, streaming application), while firmware is specialized for executing on subsystems (e.g., TEE/TA) or discrete hardware (e.g., Video decoder).

**Updating software is generally implemented via Over-The-Air (OTA) update**. It often does not depend on many partners and does not require the intervention of the SoC vendor. **The risk is relatively low for performing software updates**, and it is expected to be a process that is in place and guaranteed for several years after the product ships in the market.

**Updating firmware is also performed via OTA update**. However, it is generally done for updating more specialized components that are under the SoC vendor’s control. Those **firmware updates can be less trivial to perform**, require further Q&A campaigns, and are not necessarily as often performed on products in the field. Firmware may receive fixes but more rarely upgrades. The time window for performing firmware updates is similar to that for software and depends on the product and related support.

**Updating hardware is done on a per-development cycle**. Typically, a change that we will design now will only enter the market after a couple of years, and the in-field devices will have to live with this hardware for the rest of their lifecycle, which can be generous for some device types (e.g., over 7 years for TVs).

## Maturity and complexity

**It took a very long time to gain maturity on streaming devices**. From having large-scale adoption or a proper secure boot scheme, to TEE integration on most clients, and massive adoption of SVP. Nonetheless, we’ve just reached a stable point with relatively robust SVPs, but many of those integrations are not standardized. They are highly complex and specialized, and **we should be mindful not to make assumptions about the impact** of an otherwise deemed “simple” change, as it may reflect in much deeper consequences for some architectures. Similarly, we should pay attention to limiting the number of involved parties for implementing a change and attempt to have the smallest number of stakeholders absorb most of the complexity, rather than pushing those responsibilities to a larger number of downstream partners.

## Security functions vs security first components

Very few of the components in a streaming device have security-first responsibilities. The TEE O/S, for example, has a main function of guaranteeing security. Similarly DRM has a main function of protecting content. Those components are generally managed, designed, and integrated by teams of security professionals.

When we are adding security functions to help in mitigating a given problem or increasing the robustness of the device or the content protection scheme, we should be mindful of where this new function will be integrated, and if it is under the responsibility of a team whose core mission is toward security. **If we push security functions to components under the responsibility of non-security professionals, we are taking an extra risk** of achieving functionality with less robustness, and eventually just displace the risk instead of addressing it.

# Solution space

## Deployment timeline expectations

We are interested in both finding a so-called short-term, pragmatic, and ideally not risky workaround mitigation to DeCENC as it stands and finding a long-term proper standardized solution that would benefit the industry to safeguard against the DeCENC class of attacks entirely.

**The timeline for a workaround is likely to be in the 2-year timeframe, while adopting a proper solution for global reach is likely to be in the 5 to 10-year timeframe**.

## Mitigation presentation structure

For each mitigation as listed below, we have a summary that goes in the below details:

|  |  |
| --- | --- |
| Productization timeframe | * Short: 1-2 years * Medium: 3-5 years * Long: 5+ years |
| Level of difficulty | * Low: small number of partners, low impact on architecture, software updates * Medium: more responsible parties, firmware updates * Hard: more responsible parties, multiple updates and coordination, firmware and/or hardware updates |
| Scale of adoption | * Targeted devices * Pockets of devices or clients * Broad critical mass across clients |
| Mitigation type | * Workaround for DeCENC * Solution for class of attack |
| Backward compatibility | * Device: new and/or in-field * Catalog: new and/or existing |

Ideally, each mitigation would have sections to present or link to the technical details, present implementation considerations, adoption and impact considerations, and remaining steps and questions to answer.

## Mitigations against attack-specific using PCM

### Prohibition of PCM-like Constructs

|  |  |
| --- | --- |
| Productization timeframe | * Medium: 3-5 year |
| Level of difficulty | * Medium: more responsible parties, firmware updates |
| Scale of adoption | * Targeted devices |
| Mitigation type | * Workaround for DeCENC |
| Backward compatibility | * Device: new * Catalog: new |

#### Basics of the proposal

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**. They are known to enable attacks such as De-CENC, where decrypted bytes may be leaked through visual channels.

To address this, **a plausible workaround is to avoid using PCM-like constructs altogether**, which will remove the exploitable path. It starts by not using them while encoding the media, so the necessary steps shall be made in the encoders to avoid using PCM-like constructs. Then the device shall enforce a no PCM-like rule that will forbid the use of PCM-like constructs during the decoding operation.

#### ‘npcm’ brand approach

To standardise the prohibition of PCM-like constructs, 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 coding units 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 set the brand to signal conformance if no prohibited constructs are present.

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.

#### Implementation Considerations

To achieve the above, we need to carry the signal of not using PCM-like constructs down to the device, leveraging a secure channel. A candidate for doing so is the DRM system that could, through the license, add a flag to indicate that the associated protected video shall not use PCM-like constructs. There are maybe other options to securely carry the no PCM-like construct signal down to the device, but they should ensure the signal is authenticated and remains within the trusted domain.

Then the enforcement part will need to be carried out by a codec-aware component. A couple of options are proposed:

Option 1: The video decoder is expected to live within the TEE domain

* the decoder is a specialized component that is maintained by non-security-first experts
* it is in the TEE-domain, but may or may not have an established channel of communication with the rest of the TEE that we could leverage to signal the no PCM-like rule
* would require, at best, a firmware upgrade of video decoder firmware; this could be complex for some devices in the field. Other devices may have a full hardware video decoder, and as such, this would not be an option for in-field devices and only for upcoming devices, and adding logic in hardware needs to be carefully planned
* many different video decoders, and as many ways to integrate them inside the SVP

Option 2: a specialized trusted application (TA) running in the TEE that would be in charge of scanning the codec stream for PCM-like constructs

* needs to write a special application with a generous complex code to be codec aware and parse the stream in search of PCM-like constructs, while complex parsing of attacker controlled data is generally not recommended within TEEs as it has a high likelihood of introducing security vulnerabilities
* cost additional bandwidth for scanning
* largely compliant with all devices with a TEE, could be standardized, and updated as needed

Option 3: a specialized trusted application (TA) running in the TEE that would oversee modifying the stream to instruct the decoder that PCM-like constructs are disabled

* needs to write a special application with a complex code to be codec-aware to locate and update the right place to put that information
* cost additional bandwidth for scanning
* largely compliant with all devices with a TEE, could be standardized, and updated as needed

Other considerations w.r.t robustness

* Ideally, when a PCM-like construct is found while it was forbidden, the playback would stop. Other attempts to patch the codecs would be a second-best effort and may present another exploitable path that is not desirable. In general, we want to stop as soon as we detect a malformed stream.
* The signal of enforcement needs to be carried from the receiving entity (e.g., the DRM TA in the TEE) to the component performing the enforcement and shall not leave the TEE, nor should it be influenced by the REE
* If PCM-like constructs are found, they could either be ignored/dropped by the enforcing component, or the playback could stop abruptly

#### Adoption and impact considerations

It may be hard to harmonize support and broad adoption of this mitigation strategy across all clients of a large streaming ecosystem: option 1, with the decoder, presents challenges of a very fragmented video decoder and SVP solutions, while option 2 may present different types of challenges due to the CPU overhead required.

In addition, and by nature of this mitigation, we are removing functionality from the encoding technologies, as we are requiring not to use PCM-like constructs anymore. It is unclear how and when PCM-like constructs are useful today, and how and when PCM-like constructs will be useful tomorrow, but we are removing this as an option from the table of the encoding technologies toolbox.

Lastly, backward adoption of this protection to any existing content catalog may be complicated by the necessity to scan the existing encoded catalog and possibly re-encode and re-package any assets as needed.

#### Remaining steps and Q&A

Option 1: What is the share of video decoders that could accommodate such logic with a firmware update?

Option 2: What is the actual CPU overhead for scanning for PCM-like constructs?

Option 3: What would it take to modify the streams and disable PCM-like constructs?

### All clear/all encrypted PCM approach

|  |  |
| --- | --- |
| Productization timeframe | * Short: 1-2 years |
| Level of difficulty | * Low: small number of partners, low impact on architecture, software updates |
| Scale of adoption | * Broad critical mass across clients |
| Mitigation type | * Workaround for DeCENC |
| Backward compatibility | * Device: new and old * Catalog: new and old |

#### Basics of the proposal

As PCM-like constructs are the weak point leveraged by DeCENC, a plausible targeted mitigation is to sanitize the use of PCM-like constructs to prevent repackaging, which will remove the exploitable path.

In the DeCENC attack, the adversary uses PCM-like constructs to repackage protected content that will eventually be decrypted but not be decoded. By verifying that PCM-like constructs are either all clear or all encrypted, we would prevent a local adversary from repackaging the protected content prior to presenting it for processing (decryption, decoding, etc).

#### Technical details

The Common Encryption Specification (23001-7) mentions that only video slice data should be protected in AVC, and only video slice data shall be protected in other NAL structured video.

To simplify the design and implementation of this approach, the specification is extended to require either all video slice data in a slice NAL to be protected or to be clear, i.e. mixed protected slice data and clear slice data in one slice NAL is prohibited.

The Common Encryption Specification allows ‘BytesOfProtectedData’ to be rounded down to a multiple of 16 bytes to avoid partial cipher blocks in subsamples for ‘cenc’ AES-CTR scheme. The alignment could lead to 0 to 15 bytes of unencrypted bytes in the beginning of a slice.

We would need to scan the payload before decryption, to check whether the payload is compliant to the above requirements and to check whether a partial PCM-like construct is present in the up to 15 bytes of unencrypted bytes.

This approach is not 100% secure, especially with the pattern encryption mode. The main goal of this approach is to provide a quick and low-cost mitigation solution that can increase the adversaries' cost of utilizing DeCENC attacks efficiently on a large scale.

#### Implementation Considerations

In order to achieve the above, we need to scan for slice NAL units on the device and verify if they are either fully encrypted or fully in the clear.

An approach is to implement a specialized trusted application (TA) running in the TEE

* the logic is simple
  + move the content inside of restricted buffers first, then parse the still-encrypted content
  + if a PCM-like construct is found at the beginning of a protected video slice (not encrypted due to block alignment), ideally stop playback (alternatives can be considered but are not encouraged)
  + otherwise, if a partially protected video slice is found, ideally stop playback (alternatives can be considered, but are not encouraged)
  + otherwise, proceed as usual (decrypt, then decode, etc)
* cost additional bandwidth for scanning
* largely compliant with all devices with a TEE, could be standardized, and updated as needed

It should be noted that adding a “codec-aware” software stack for performing the scan of PCM-like constructs introduces a level of complexity inside the TEE that is not desirable. This complexity should at least be well isolated into a dedicated Trusted Application to prevent introducing more risk than it solves.

#### Adoption and impact considerations

This solution has a CPU overhead, and perhaps not all clients will be able to spare the required cycles for implementing this mitigation. As such, the performance impact shall be evaluated.

This option removes any dependency on encoding technologies, which can remain free to use the entire codec specifications. Instead, it targets the detection of abuse of PCM-like constructs. By doing so, we could have a software-based mitigation that could be broadly adopted by a variety of vendors on their products.

Development could be centralized, and a reference code could be offered to partners for evaluation and integration.

#### Remaining steps and Q&A

What is the CPU overhead for scanning for all clear/all encrypted PCM-like constructs?

Is this compatible with all forms of SVPs?

What is the share of compatibility in the field?

## Integrity protection

|  |  |
| --- | --- |
| Productization timeframe | * Medium to Long: 3 to 5+ years |
| Level of difficulty | * Hard: more responsible parties, multiple updates and coordination, firmware and/or hardware updates |
| Scale of adoption | * Broad critical mass across clients |
| Mitigation type | * Solution for class of attack |
| Backward compatibility | * Device: new * Catalog: new |

### Basics of the proposal

Adding integrity protection to the payload would eradicate the entire class of attacks abusing codec complexity. It has the potential to safeguard today and future codecs, the distribution, and playback of protected content: the processed bitstream by the on-device decoder has not been tampered with, it is as intended and prepared in the backend.

This adds a whole layer of security guarantees over the existing protection layer, which leverages DRM and content encryption.

For integrity protection to be effective against codec abuses, it should cover the entirety of the video payload: protected and unprotected ranges.

#### Limits and weaknesses

It should also be noted that this is indeed a proposed authentication scheme using a MAC, not a signature. By construction, the streaming client devices will be in possession of the integrity key itself, and if abused/leaked, it could be used to forge payloads that will be accepted as valid.

### Design considerations

Encrypt-then-MAC is the most common way to authenticate data. This approach may allow for the use of authentication technologies that are under development for other media authentication purposes, as is the case for MAC-and-encrypt. However, whether encrypt-then-MAC or MAC-and-encrypt is selected depends on what the application is and what a security analysis of that application recommends.

#### Processing order

**Encrypt-then-MAC**

This strategy consists of first performing (potentially partial) encryption and then computing the MAC over the (potentially partially) encrypted payload. Phrased differently, the MAC is computed over the payload after encryption on the server side, and before decryption on the client side.

Pros:

* This is generally considered the best practice to avoid creating oracles.

Cons:

* Needs to authenticate the encryption parameters, which adds complexity for both current and future implementations. These encryption parameters should include at least the encryption mode, the encryption mode parameters if any, the key material including the encryption key itself and the IV if any, and the ranges of protected and unprotected data. Missing any one of these parameters can open a path to an attack.

Covering such parameters is not part of usual cryptographic modeling. For this reason, we did not perform a cryptographic analysis.

**MAC-and-encrypt**

This strategy consists of independently computing the MAC over the payload and performing (potentially partial) encryption. Phrased differently, the MAC is computed over the payload before encryption on the server side, and after decryption on the client side. The MAC itself is not part of the payload when encryption is performed.

Pros:

* What is authenticated is what enters the decoder, narrowing the window where it can be tampered with to mount an attack. This follows the TOCTOU strategy.
* In this order, the verification is done after decryption. The integrity verification function should be implemented in the DRM TA as advocated in a later section. Although this is discouraged, the integrity verification function could also be implemented within the decoder or anywhere between the DRM TA and the decoder as long as no parsing is performed on the data between decryption and integrity verification.

Cons:

* Not the default, must be careful not to create oracles.

We performed a cryptographic analysis and did not find any security flaw when using MAC-and-Encrypt with CBCS encryption (AES-CBC with skip pattern 1:9).

#### Integrity scope

For security and cost reason, it would be better to avoid explicitly indicating ranges of covered data.

#### Maximum length

Covering the entire stream for integrity protection shall be realistic and possibly adopted at a very large scale on devices. Several constraints for adoption are discussed hereafter.

Devices have finite resources. TEE and secure environments might be even more constrained. To guarantee the device's ability to perform the integrity verification step, the maximum length of the payload that is covered by a single verification operation must be limited. It is therefore helpful to split the data covered by integrity into different chunks. Furthermore, this process allows decoders to more efficiently process content: decoding can start as soon as a single chunk is decrypted and authenticated instead of having to wait for the decryption and availability of the data from an entire frame, which can be potentially very large.

We don’t expect a noticeable performance difference when computing several MACs for a large frame compared to only one MAC. Since the total amount of data to cover is the same, the overhead should be negligible if each individual MAC covers several kilobytes of data.

Fixed length provides a reasonable way to ensure devices will be built and designed to accommodate the required resources needed for this integrity verification step. Nonetheless, it also provides an arbitrary number that may work better for some device makers and has other implications, as the number of operations per second will scale up as we increase the bandwidth. For example, on a very high bitrate of 50Mbps, we might be doing over 380 “MAC” verifications per second with a 16K maximum size. This should be evaluated on a variety of streaming devices and bandwidths.

There are a couple of approaches with pros and cons that should be discussed

* Determine a fixed length and go with it, for e.g., 16K
* Determine a series of fixed lengths and produce recommendations based on resolution and/or bandwidth, for e.g., 4K, 16K, 32K, 64K, with a recommendation of 16K for 4K content.

#### Chaining

In case several MACs cover one unit of data, it is necessary to define a chaining mode to prevent an attacker to shuffle the chunks covered by each individual MAC. A simple approach is to prepend the first chunk with a block encoding the length of the frame, and then prepend each chunk with the MAC of the previous chunk, when computing the MACs. It should be proven that the exact mode to be used is secure.

#### Integrity algorithms

A variety of algorithms and modes can be used to implement the integrity tag computation.

Most common ones are based on the AES block cipher or on the SHA hash functions. Some of them are benchmarked below.

### Other design considerations

#### Integration with DRM systems

Integrity verification is a security function, and DRM systems are a good candidate for hosting that function. In addition, there is a fragmentation piece to consider: there are a handful of 1-digit numbers of DRMs, while there are 2-digit numbers of SoC vendors, 3-digit numbers of manufacturers, and so forth. It is in everyone’s interest that complexity remains in the hands of a limited number of entities.

**Location of the integrity verification function**

Depending on what order we choose for integrity verification and decryption, we might be more inclined to perform the verification in different places. For example, the DRM is the entity that acquires the license, then drives the decryption. If integrity verification shall happen prior to decryption, it then needs to happen between license acquisition and decryption, and depending on the system, this will have design consequences on the SVP pipeline. While if integrity verification shall happen after decryption, it leaves the option for SoC designers to place the integrity verification either in the DRM, in the video decoder, or anywhere in between, for as long as it remains in the TEE domain.

When integrity verification is performed after decryption, there should be no processing of decrypted data until its integrity is verified. Any error condition or timing difference may leak information about the value of decrypted data and could open the way to an attack. To be clear: in the MAC-and-encrypt approach, the decryption process and the integrity verification process should be tightly coupled together. This is another argument for having to perform the integrity verification in the DRM. More generally complex parsing of attacker controlled data is generally not recommended within TEEs, as it has a high likelihood of introducing security vulnerabilities

Ideally, the DRM system will be designed to allow a call for a porting interface, which would be in charge of performing the required steps to ensure integrity verification is performed or will be performed down the road.

**DRM licenses**

DRM licenses would carry the signal for enforcing the use of integrity verification; licenses will also carry the actual authentication key and the ID to refer to it.

It is assumed that a unique authentication key can be assigned to any content. Content service providers can, similarly to how they encrypt content, decide on the uniqueness of that key and applicability across resolutions. Logically, the key must be paired with the encryption key for the mitigation to be effective.

For identifying the authentication key, we could use a different key ID for the encryption keys and the integrity verification key, or use a single Key ID that will bind both encryption and integrity verification keys.

**Failure behavior**

It is necessary to halt playback as soon as a presented stream violates the requirement from the license. Here, if the integrity verification fails, no further treatment should be performed. Parsing data while the integrity check failed might leak information and open the way to an attack. In particular, no parsing should occur between the decryption process and the integrity process in the MAC-and-encrypt approach.

#### Concurrent SVPs

It took a long time to get SVPs to be mature, and we are still working on enabling secure concurrent SVPs, with the possibility of running in-clear video concurrent to SVP/DRMed videos. Adding integrity protection as designated by the content owner through the license leads us to a situation where we will have to support multiple concurrent, some going in the clear, others through the SVP, and some of which may have integrity protection enabled, while others may not. This is possible, but it will come with the cost of architecture evolution and verification of conformance.

### Adoption and impact considerations

#### Overheads

Integrity protection adds overhead in multiple places:

* At encoding time, CPU overhead
* At distribution, bandwidth overhead
* At playback, CPU and latency overhead

We benchmarked a few algorithms that we believe are widely available on devices and can be used to compute integrity tags.

In particular, we benchmarked:

* MAC modes based on the AES block cipher, and
* HMAC based on the SHA2 family of hash functions.

Different factors must be considered when comparing such algorithms including:

* existing devices support,
* the size of the additional data (e.g. integrity tag and potentially an IV),
* the speed.

Speed benchmarks were conducted using OpenSSL version 3.4.1 from homebrew on an Apple M1 Max laptop for different sizes of data by running the command:

openssl speed -decrypt -evp <algo> -seconds 10 -bytes <size>

This relies on hardware instructions when available. They are given in gigabytes per seconds.

In case an algorithm was not directly exposed, the speed measurements were extrapolated from other speed measurements.

**AES-GCM**

AES-GCM is an authenticated encryption mode (both encryption and authentication).

It can not readily be used to add integrity only together with another arbitrary mode of encryption unless used in GMAC mode as described in a later section.

Using it both for encryption and authentication in the scope of Common Encryption is challenging without deviating from its standardized form, in particular when supporting partial/skip encryption, and should not be attempted without a thorough cryptographic analysis.

AES-GCM, being the standard mode for authenticated encryption, is expected to be widely supported in hardware.

The size of one integrity tag is 16 bytes for the tag and 12 additional bytes for the IV (shared with encryption).

We measured the following speed performance in gigabytes per second for different sizes of data using aes-128-gcm:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 16B | 64B | 256B | 1kB | 8kB | 16kB | 1MB | 8MB | 16MB |
| 0.10 | 0.39 | 1.25 | 3.56 | 7.08 | 7.66 | 8.35 | 8.33 | 8.35 |

**GMAC**

GMAC can be described as AES-GCM with only associated data and no data to be encrypted.

GMAC, being the authentication part of AES-GCM, is expected to benefit from a similarly wide hardware support.

The size of one integrity tag is 16 bytes for the tag and 12 additional bytes for the IV.

We extrapolated the following speed performance in gigabytes per second for different sizes of data by subtracting those for aes-128-ctr from those for aes-128-gcm:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 16B | 64B | 256B | 1kB | 8kB | 16kB | 1MB | 8MB | 16MB |
| 0.12 | 0.43 | 1.49 | 5.30 | 17.25 | 20.81 | 26.21 | 27.10 | 27.18 |

**CMAC**

The size of one integrity tag is 16 bytes.

We measured the following speed performance in gigabytes per second for different sizes of data using aes-128-cmac:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 16B | 64B | 256B | 1kB | 8kB | 16kB | 1MB | 8MB | 16MB |
| 0.17 | 0.45 | 1.10 | 1.39 | 1.51 | 1.52 | 1.53 | 1.52 | 1.51 |

This specific implementation seems not to use hardware instructions. We, therefore, also provide the speed measurements for aes-128-cbc, as shown below, that uses hardware acceleration and should reflect better on what performance a hardware oriented implementation would provide:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 16B | 64B | 256B | 1kB | 8kB | 16kB | 1MB | 8MB | 16MB |
| 0.70 | 4.26 | 8.91 | 12.55 | 14.01 | 14.18 | 14.32 | 14.06 | 13.96 |

**HMAC**

HMAC, being a standardized authentication mechanism, is expected to be widely supported in hardware.

The size of one integrity tag is at least 32 bytes using SHA256.

We measured the following speed performance in gigabytes per second for different sizes of data using hmac-sha256:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 16B | 64B | 256B | 1kB | 8kB | 16kB | 1MB | 8MB | 16MB |
| 0.08 | 0.31 | 0.92 | 1.70 | 2.22 | 2.30 | 2.35 | 2.35 | 2.35 |

#### HW DRM vs SW DRM considerations

Mitigating for DeCENC attacks in SW DRM will be even more challenging, as per the attack promise, the local adversary is already in control of the REE-domain, where the SW DRM lives.

Nonetheless, there are additive values in enabling integrity verification that would help SW DRM overall hardening.

Enabling integrity verification may not help SW DRM much, as it can be relatively easily bypassed. Moreover, it could hurt performance and expose additional attacking surfaces.

#### Interactions with bitstream manipulation use

Adding integrity verification will, by nature, impede the ability to enable use cases dependent on it. For example, 360 views with tiles merge or modification of the bitstream at the edge for implementing novel watermarking solutions will be limited.

Those use cases can still exist and be deployed, but can’t benefit from an integrity-protected video profile.

### Remaining steps and Q&A

What is the share of availability of crypto algorithms across different clients? (mobile, vs TV, vs OTT)

What is the performance of crypto algorithms on a variety of different clients? (mobile, vs TV, vs OTT)

What is the acceptable level of overhead we are willing to take (CPU, bandwidth)?

# Call to the community

## Representation of perspectives

The streaming ecosystem is broad, including device stakeholders, content studios, streaming services, and so forth. We all have perspectives on a slice of the ecosystem, and we only come stronger together with a higher, big-picture view.

We need to have a healthy representation of partners in the streaming device ecosystem, bringing expertise in content protection, device security, encoding technologies, etc., for the broad class of affected devices.

In particular, in the past workshop, we lacked representation of SoC vendors, and we should encourage their participation in the group.

## Robustness analysis of mitigations

We invite the community to provide robustness analyses of mitigations in support of or against proposed schemes. Below is a list of performed analyses and their outcomes.

### Integrity protection - MAC-and-Encrypt

Apple did a cryptographic analysis of adding integrity protection with a MAC-and-Encrypt order and using CBCS (AES-CBC with 1:9 encryption pattern). The analysis did not find critical problems or security flaws that would prevent the adoption of such a mitigation scheme.

Apple did not analyse other protection schemes, for e.g., covering AES-CTR.