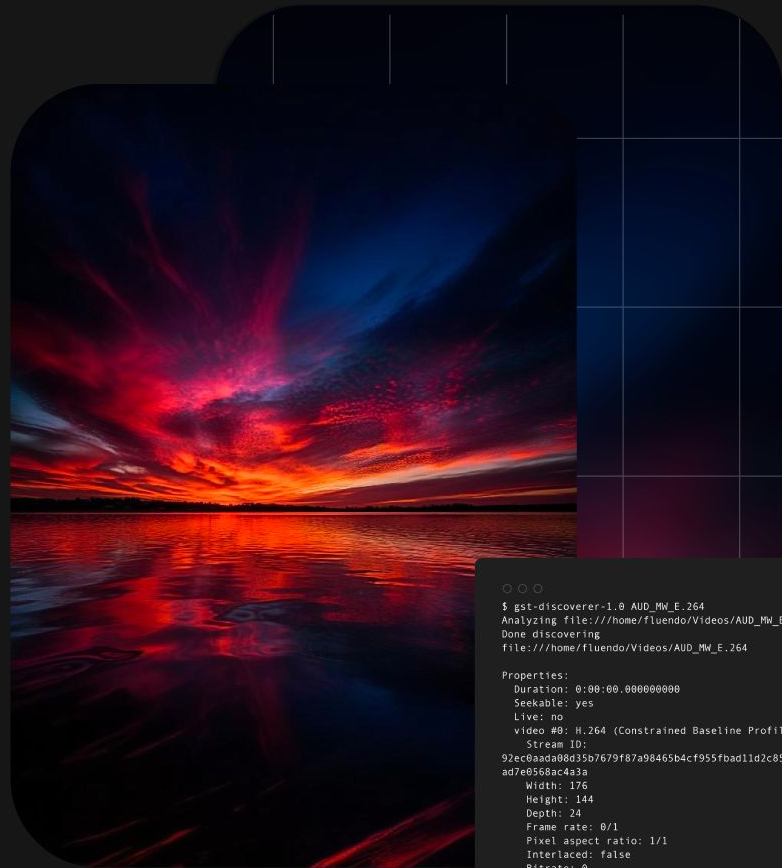




Digitally Signed Content & C2PA in GStreamer

GStreamer talk



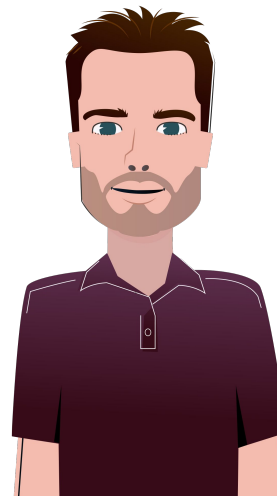
```
$ gst-discoverer-1.0 AUD_MW_E.264
Analyzing file:///home/fluendo/Videos/AUD_MW_E.264
Done discovering
file:///home/fluendo/Videos/AUD_MW_E.264

Properties:
  Duration: 0:00:00.000000000
  Seekable: yes
  Live: no
  video #0: H.264 (Constrained Baseline Profile)
    Stream ID:
    92ec0aada08d35b7679f87a90465b4cf955fbad11d2c8535ec
    ad7e0560ac4a3a
    Width: 176
    Height: 144
    Depth: 24
    Frame rate: 0/1
    Pixel aspect ratio: 1/1
    Interlaced: false
    Bitrate: 0
    Max bitrate: 0
```

Diego Nieto

■ Diego Nieto

- Senior Multimedia Engineer at Fluendo (dnieto@fluendo.com)
- Some GStreamer work:
 - VMAF
 - Video Encoder Tools
 - H.266 Alpha layer
 - CUDA video processing within NVIDIA Jetson



DSC & C2PA agenda

Introduction

How Digitally Signed Content works

Implementing DSC support in GStreamer

Implementing C2PA-DSC support in GStreamer

Demo: Signing and verifying video content

Introduction

Introduction I



Introduction I



Introduction I



Introduction II

- As AI technology advances, it is becoming increasingly difficult to determine whether video content is authentic or fake.
- Users are losing trust in the information that they consume.
- Malicious players may not label their content at all or remove such marking.
- To overcome this problem, JVET has proposed a mechanism for trustworthy signing and authentication of coded video data at an elementary stream level in their JVET-AN1019-v1 contribution. It was called Digitally Signed Content.

Introduction III

- The method operates similarly to the process of digitally signing documents submitted to government authorities.
- Digitally Signed Content is compatible with:
 - random access
 - scalable coding
- The proposed DSC signaling is integrated into the ITU H.274 (VSEI) standard, which is designed for reusability across diverse video codecs.

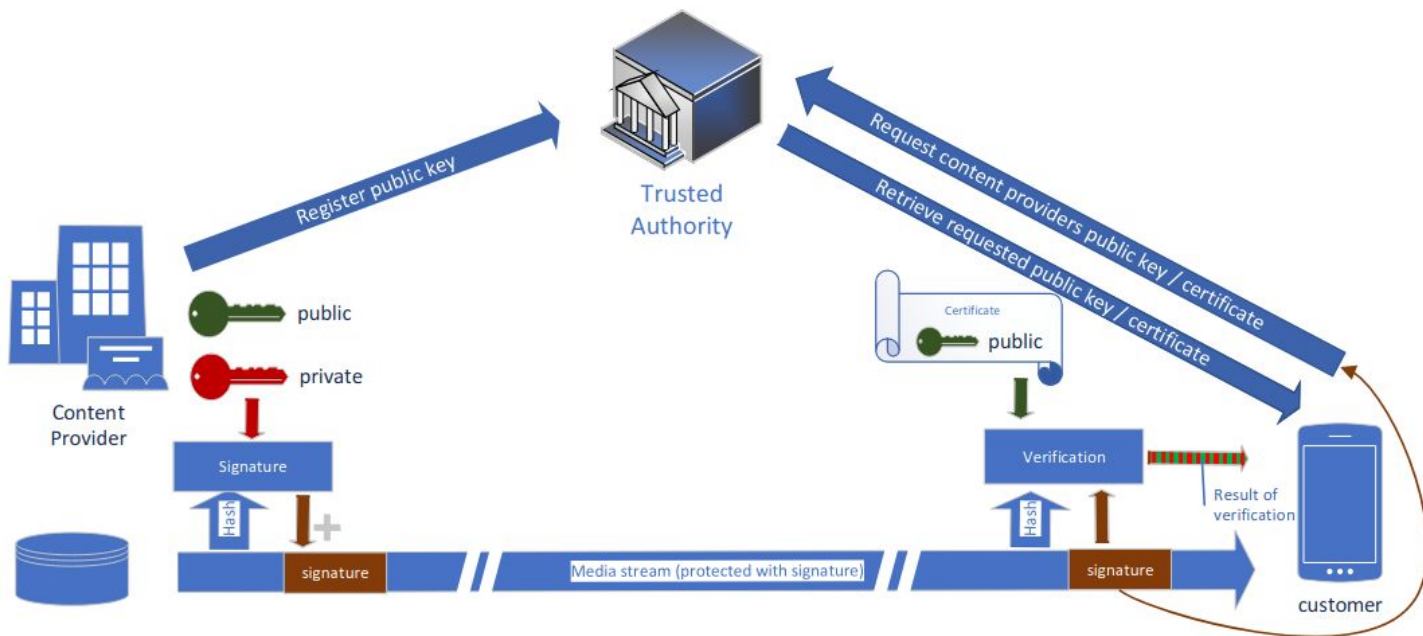
Fraunhofer & Fluendo collaboration

- Fraunhofer HHI implemented Digitally Signed Content in the reference encoder and decoder for H.265 and H.266:
 - VTM
 - HM
- Fluendo implemented in GStreamer Digitally Signed Content making it compatible with the reference software and C2PA.



How DSC works

Global picture



Definition of a Substream

- A substream relates to a specific flow of the stream.
- Supports:
 - temporal scalability, where some temporal layers can be dropped.
 - multi-view coding, where pictures can be displayed for a specific view only.
- Each substream can be verified independently from substreams which, in a given ordering, are either of higher order or are unrelated to it with respect to the ordering.
- Verification of a substream of higher order is only possible in combination with the verification of substreams of lower order.
- For now only single layer (substream) is supported in both reference encoders/decoders and GStreamer.

DSC in H.266/VVC & H.265/HEVC

- Authentication and verification of video content through new Supplemental Enhancement Information (SEI) messages:
 - DSCI, Initialization SEI: will find message at the beginning of the substream, i.e. in the first frame of the substream. It will come with the DSCS message afterwards.
 - DSCS, Selection SEI: will find this one once per frame.
 - DSCV, Verification SEI: will find this one at the end of the substream. The DSCS message will be before of DSCV message.
- These SEI messages enable attaching cryptographic signatures to **flexible chunks of data of a video stream at the NAL unit level**.

Digitally Signed Content workflow simplified

How DSC works

DSC Substream — Message Placement

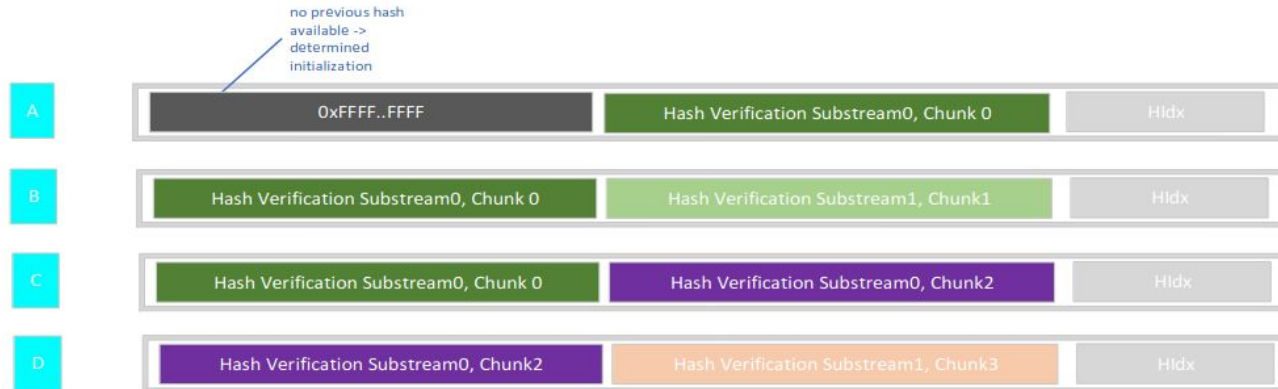
 DSCI (first AU)

 DSCS (every AU)

 DSCV (last AU)



How DSC works



Recommended Chunk Dependencies

The three SEI messages – DSCI (Initialization)

The DSCI SEI message indicates the beginning of signed content and provides the initialization parameters for the entire signing process:

- **dsci_id**: an identifying number of the verification system.
- **dsci_hash_method_type**: indicates the secure hash algorithm used (SHA-1, SHA-224, SHA-256, SHA-384, SHA-512).
- **dsci_key_retrieval_mode_idc**: specifies how to retrieve the public key (trust record or certificate).
- **dsci_use_key_register_idx_flag**: indicates whether a key register index is used.
- **dsci_key_register_idx**: index specifying the certificate in a certificate register (when applicable).
- **dsci_content_uuid_present_flag**: indicates whether a content UUID is present.
- **dsci_content_uuid**: unique identifier for the video content (UUID).
- **dsci_num_verification_substreams_minus1**: number of verification substreams minus 1.
- **dsci_ref_substream_flag**: flags indicating dependencies between substreams.
- **dsci_vss_implicit_association_mode_flag**: verification substream implicit association mode.
- **dsci_signed_content_start_flag**: indicates the start of signed content.
- **dsci_sei_signing_flag**: indicates whether SEI messages are signed.
- **dsci_key_source_uri**: URI reference to the content creator's public key certificate or trust record.

The three SEI messages – DSCS (Selection)

The DSCS SEI message indicates which verification substream a particular picture belongs to:

- **dscs_id**: the verification system ID (matches `dsci\_id`).
- **dscs_verification_substream_id**: the ID of the verification substream that this picture is part of.

The three SEI messages – DSCV (Verification)

The DSCV SEI message is transmitted for each of the verification substreams and contains the actual cryptographic signature:

- **dscv_id**: the verification system ID (matches `dsci\_id`).
- **dscv_verification_substream_id**: the ID of the verification substream being signed.
- **dscv_signature_length_in_octets_minus1**: length of the signature data minus 1.
- **dscv_signature**: the digital signature of the substream data.
- **dscv_signed_content_end_flag**: indicates the end of signed content.

Protection against temporal attacks

- A critical feature of the DSC design is protection against attacks by removal, insertion, or shuffling data chunks in the protected data stream.
- To guarantee that decoders can verify the continuity between different temporal segments of the coded video data, the hash value of a preceding temporal segment is also incorporated in the digital signature of a current temporal segment (kind of blockchain schema)
- Consequently, a joint digital signature of both a current and its previous temporal segment is constructed.



Signing workflow I

What do we hash?

- The NAL units extracted from each video buffer (using the custom internal NAL parser).
- Only signable NAL data is selected (e.g., excluding non-signable SEI, filler, etc.).
- Each buffer's signable NAL units are added to the current substream.

When do we hash?

- On every buffer (video/x-h266), AU.
- For each buffer, the NAL units are parsed and immediately added to the substream manager for hashing.
- Hashing occurs incrementally as each buffer in the substream is received.

Signing workflow II

What do we sign?

- We sign the “data packet” created at the end of each chunk (GOP in our case).
- The data packet is constructed by the DscSubstreamManager from all hashed NAL units in the substream.

The data packet is a binary blob containing:

- The hash method identifier (u8): Specifies which hash algorithm was used (e.g., SHA-256).
- The content UUID ([u8; 16]): Uniquely identifies the video content. (Optional field if no C2PA!)
- The concatenated hashes of all signable NAL units in the substream (Vec<u8>).
- Optionally, other metadata required for verification (e.g., substream ID).

Signing workflow III

When do we sign?

- When the last buffer in the substream (GOP) is processed (i.e., `buffers_in_substream >= substream_length`).
- At this point, all NAL data for the substream has been collected and hashed.
- The substream manager creates a data packet from the hashed NALs, which is then signed using the loaded private key.

Signing workflow IV

When do we create each SEI message?

- DSC Initialization SEI: Added to the first buffer of each substream (GOP). Contains hash method, content UUID, and public key URI.
- DSC Selection SEI: Added to every buffer in the substream, indicating the substream ID.
- DSC Verification SEI: Added only to the last buffer of the substream, containing the signature for the substream's data packet.

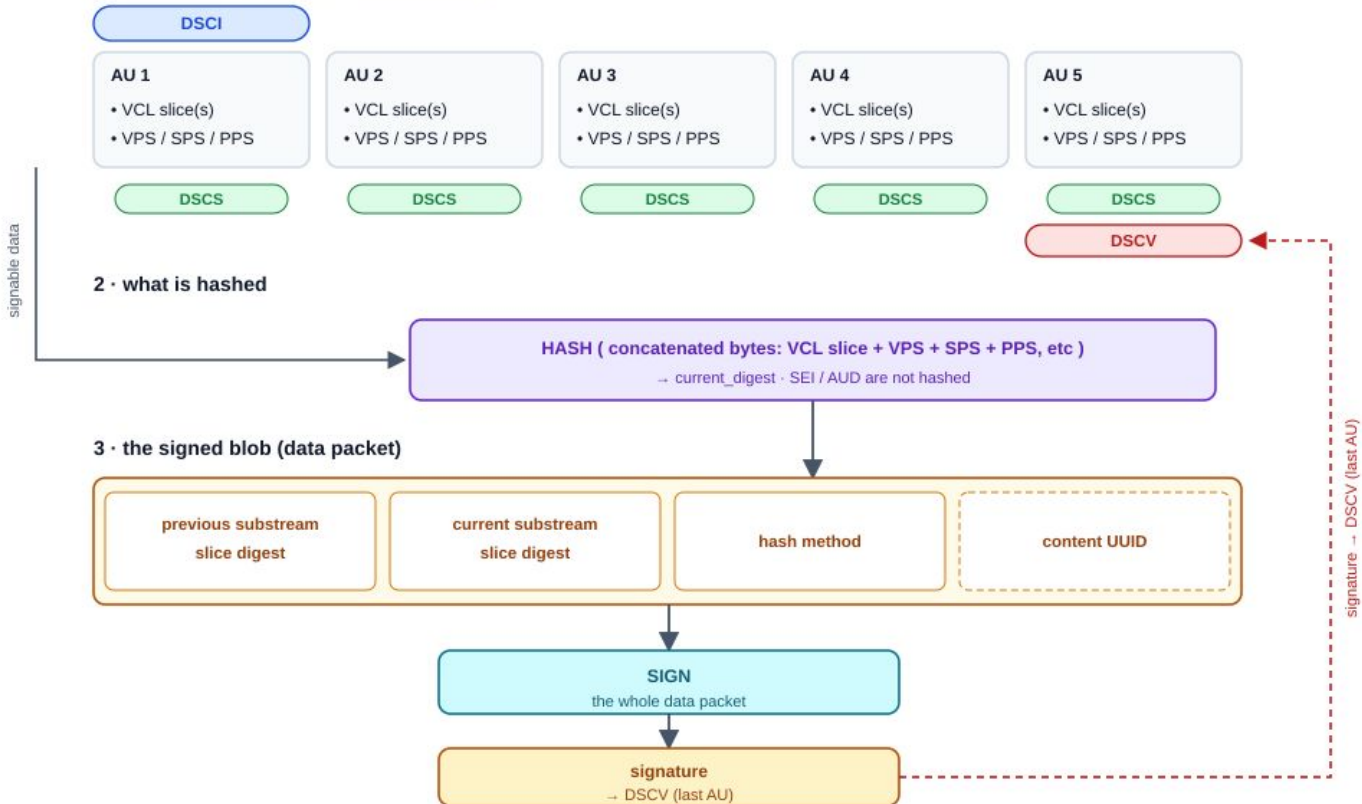
Signing workflow top level

- On first buffer of substream chunk:
 - Add DSC Initialization SEI metadata.
 - Update substream manager.
- For every buffer:
 - Add DSC Selection SEI metadata.
 - Parse and hash signable NAL units.
- On last buffer of substream chunk:
 - Create data packet from hashed NALs.
 - Sign the data packet with the private key.
 - Add DSC Verification SEI metadata with the signature.
 - Reset substream state for next GOP.

DSC — Signing

■ DSCI ■ DSCS ■ DSCV

1 · substream — a GOP of N access units





Verification workflow I

What do we verify?

- The signature attached to the last buffer of each chunk (GOP) via DSC Verification SEI metadata.
- The signature is checked against the reconstructed data packet (same structure as signed during encoding)

When do we verify?

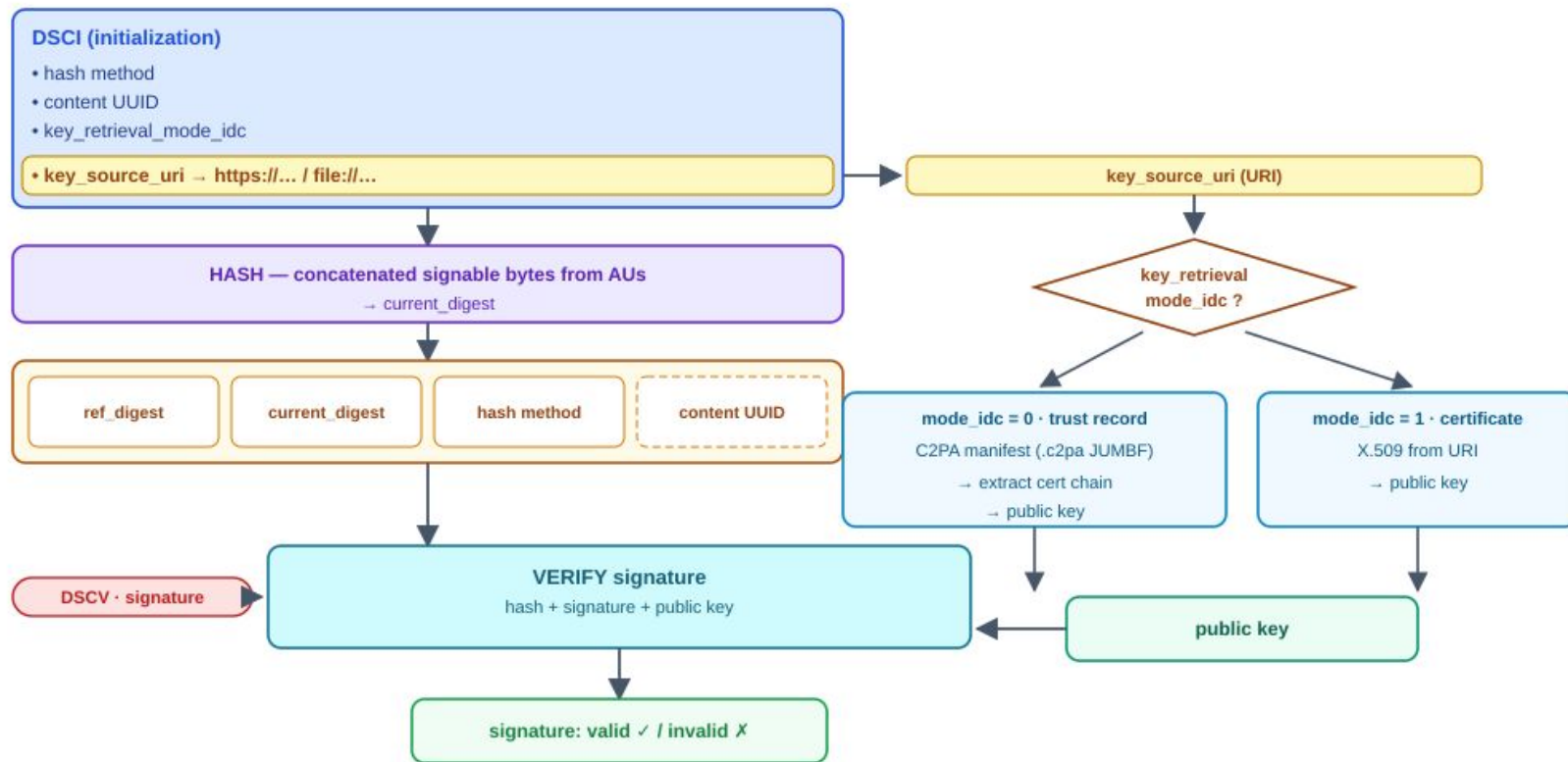
- On every buffer, the verifier checks for DSC Verification SEI metadata.
- Verification is triggered only when such metadata is present (i.e., on the last buffer of a substream).
- For other buffers, NAL units are accumulated for later verification.

Verification workflow top level

- On buffer with DSC Initialization SEI:
 - Parse hash method, content UUID, and public key URI.
 - Initialize or reset the substream manager.
- On every buffer:
 - Parse and accumulate signable NAL units for the current substream.
- On buffer with DSC Verification SEI:
 - Reconstruct the data packet from accumulated NALs.
 - Load the public key from the key store.
 - Verify the signature using the public key and hash method.
 - Post verification result (success/failure) as a GStreamer message.
 - Optionally, fail the pipeline if verification fails (configurable).

DSC — Verification

□ DSCI
 □ DSCS
 □ DSCV



Implementing DSC support in GStreamer

Support Digitally Signed Content video (DSC) in GStreamer

☒ Open ☐ Issue created 8 months ago by **Diego Nieto**

Digitally Signed Content allows to verify the authenticity of the videos.

They are mechanisms for trustworthy authentication and verification of video content have recently been developed by JVET for inclusion into these video coding standards. This has been realized by three new supplemental enhancement information (SEI) messages which enable to attach cryptographic signatures to flexible chunks of data of a video stream at the network abstraction layer (NAL) unit level.

Paper: https://www.hhi.fraunhofer.de/fileadmin/Events/2025/IBC_2025/IBC2025PaperAuthentication_HHI.pdf

JVET Specs: https://www.jvet-experts.org/doc_end_user/documents/40_Geneva/wg11/JVET-AN1019-v1.zip

References implementations:

- DSC implementation in VVC VTM: https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM/-/releases/VTM-23.13
- DSC implementation in HM: https://vcgit.hhi.fraunhofer.de/jvet/HM/-/merge_requests/110

To support them in GStreamer we would need to:

- ☒ Merge the support of parsing and inserting DSC SEI messages in H.266/VVC bitstreams:
 - MR: [!10435 \(merged\)](#)
- ☒ Merge a new `h266seiinserter` on the top of [!10367 \(merged\)](#) supporting DSC SEI messages: [!10514 \(merged\)](#)
- ☐ Merge a `dsc` plugin that contains:
 - `dscsigner` element that signs and generates the DSC metas of a given coded bitstream
 - `dscverifier` element that reads the DSC metas and verifies the authenticity of a given coded bitstream
 - Support for
 - ☒ H.266/VVC
 - ☒ H.265/HEVC
 - ☐ H.264/AVC
 - MR: [gst-plugins-rs!2845](#)
- ☒ Merge GIR files for GStreamer bindings:
 - MR: [gir-files-rs!103 \(closed\)](#)
- ☒ Merge the bindings for the DSC metas in gstreamer-rs:
 - MR [gstreamer-rs!1907 \(merged\)](#)
- ☒ Merge the support of parsing and inserting DSC SEI messages in H.265/HEVC:
 - MR: [!11755 \(merged\)](#)

A Dockerfile is provided here <https://github.com/fluendo/dsc> to facilitate the cross-validation with the reference encoders and decoders.

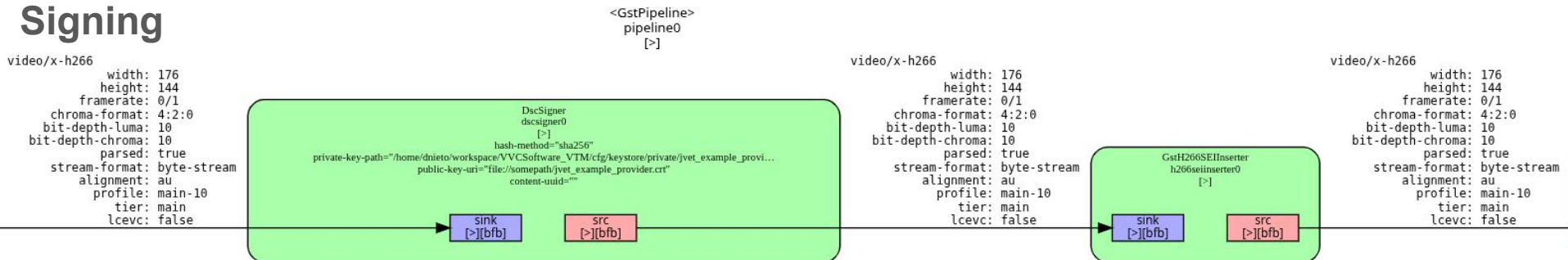
#4841

DSC examples and docker
available on:

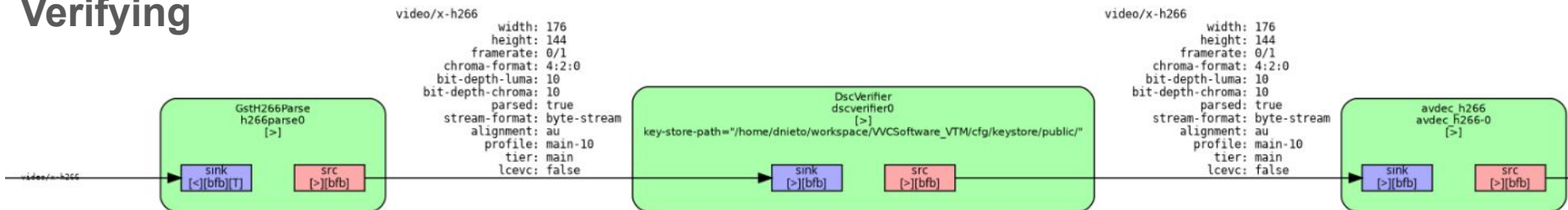
<https://github.com/fluendo/dsc>

DOT examples

Signing



Verifying

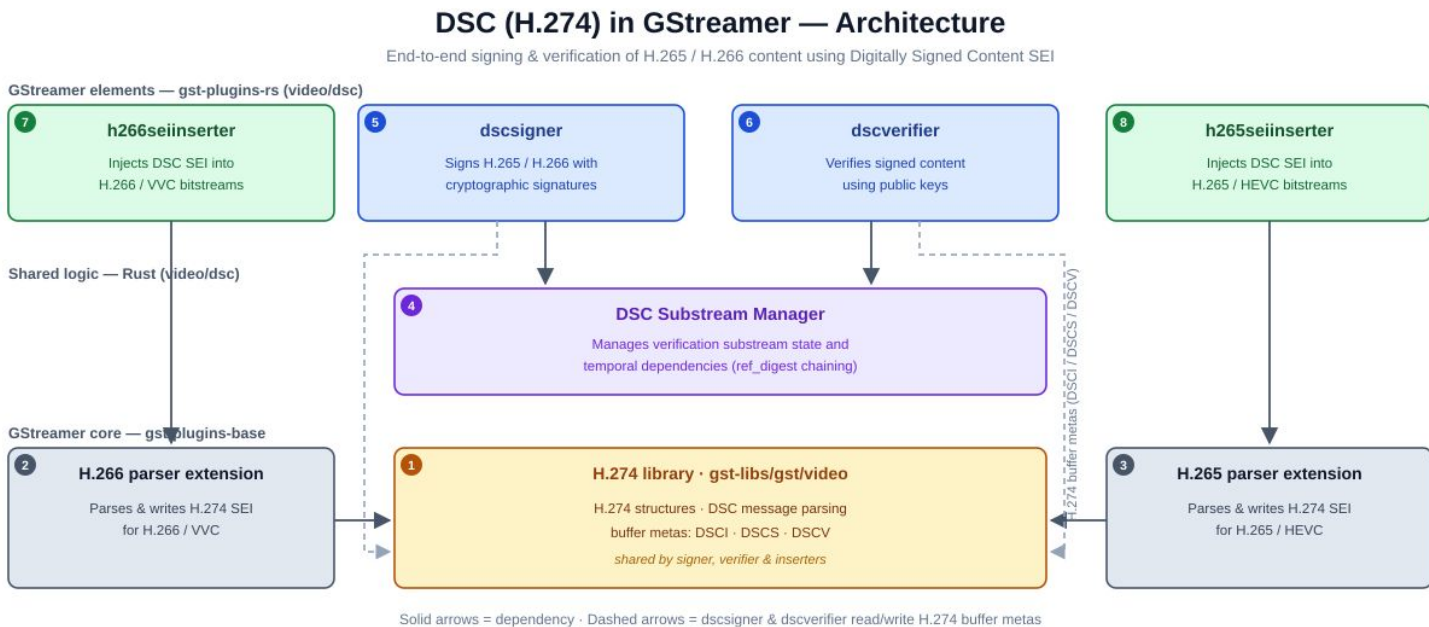


Architecture Overview

Our implementation consists of several components working together:

1. **H.274:** Adds H.274 structures and parses its messages, including DSC ones.
2. **H.266 parser extension:** Implements the parsing and writing of H.274 SEI messages for H.266
3. **H.265 parser extension:** Implements the parsing and writing of H.274 SEI messages for H.265
4. **DSC Substream Manager:** Manages verification substream state and temporal dependencies. Used by `dscsigner` and `dscverifier`.
5. **DSC Signer Element** (`dscsigner`): Signs H.266/VVC streams with cryptographic signatures.
6. **DSC Verifier Element** (`dscverifier`): Verifies signed H.266/VVC content using public keys.
7. **H.266 SEI Inserter Element** (`h266seiinserter`): Injects DSC SEI messages into H.266/VVC bitstreams.
8. **H.265 SEI Inserter support** (`h265seiinserter`): Injects DSC SEI messages into H.265/HEVC bitstreams.

Architecture Overview



Parsing DSC SEI messages I

Since the ITU H.274 specification is designed to be codec-agnostic and reusable across H.266/VVC, H.265/HEVC, and H.264/AVC, we implemented a **common H.274 parser infrastructure** in GStreamer that can be shared across all H.26x codec parsers.

Introducing `gsth274` and `gsth274parser`

To provide a solid foundation for DSC implementation, we added two new libraries to GStreamer:

- **`gsth274` (in `gst-plugins-base`):** Defines core data structures for H.274 SEI messages following the ITU-T H.274 (VSEI) standard. Provides codec-independent type definitions and memory management functions for DSC structures.
- **`gsth274parser` (in `gst-plugins-bad`):** Implements parsing and writing logic for H.274 SEI messages. Handles bitstream operations using GStreamer's NAL reader/writer utilities to process DSCI, DSCS, and DSCV messages.

Parsing DSC SEI messages II

This architectural approach provides several key advantages:

- **standardized data structures** across all codecs,
- **simplified maintenance** with centralized bug fixes and improvements.

The H.266 parser (`h266parse`) automatically detects DSC SEI messages and uses the common H.274 parser to extract the relevant information for downstream processing.

The H.265 parser (`h265parse`) automatically detects DSC SEI messages and uses the common H.274 parser to extract the relevant information for downstream processing.

This makes it straightforward to add DSC support to H.264 parsers in the future (they can simply call the same `gst_h274_parser` functions when processing SEI NAL units).

Handling metadata with GstMeta

Once parsed, the cryptographic signatures, substream dependencies, and UUIDs need to travel through the GStreamer pipeline alongside the video data. We created three custom GstMeta structures to attach DSC information to GstBuffer objects, one for each SEI message type:

- *GstVideoDSCInitializationMeta*
- *GstVideoDSCSelectionMeta*
- *GstVideoDSCVerificationMeta*

They are available in `gstvideodscmeta.c`

DSC Signer element

We developed the `dscsigner` element to sign video streams with DSC. They key properties of `dscsigner`:

- `private-key-path`: Path to the RSA private key file
- `public-key-uri`: URI of the corresponding public key certificate
- `substream-length`: Number of frames per chunk
- `hash-method`: Hash algorithm (SHA-256, SHA-384, SHA-512)

DSC Verifier element

The `dscverifier` element is responsible for verifying signed content. The key properties of `dscverifier`:

- `key-store-path`: Directory containing trusted public key certificates
- `fail-on-verification-error`: Whether to stop the pipeline on verification failure (default: true)

The verifier posts element messages on the bus when verification succeeds or fails, allowing applications to react accordingly. Applications can listen for `dsc-verification-result` messages containing verification status, data packet size, signature size, and error details if applicable.

SEI insertion element

SEI Insertion Element

To support encoders that don't natively support DSC, we created the `h266seiinsertter` element that can inject DSC SEI messages into existing H.266/VVC bitstreams as well as other H.274 SEI messages already supported

For now we did not implement reordering (it would be needed for closedcaptions in the future)

Implementing C2PA-DSC support in GStreamer

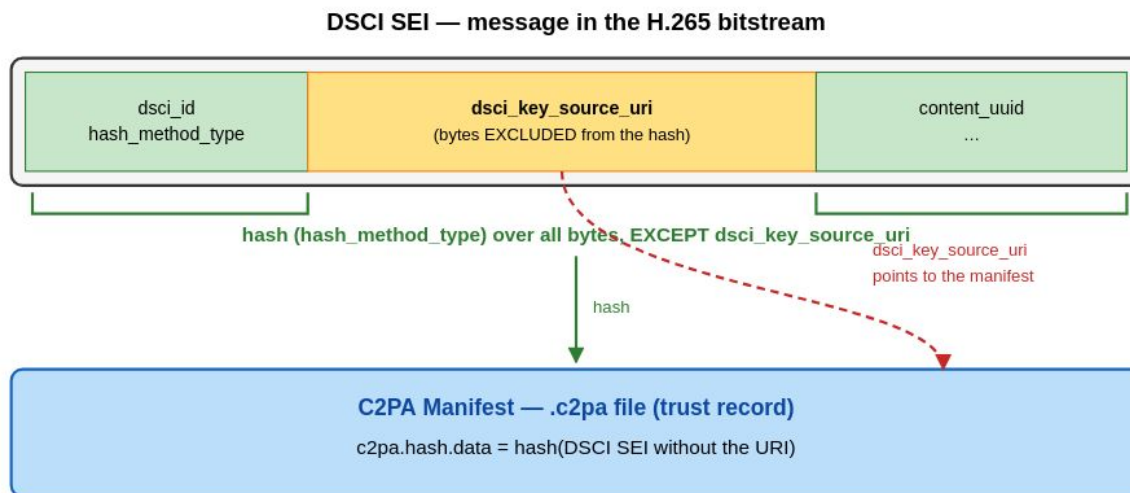
C2PA

- Coalition for Content Provenance and Authenticity
- It supports multiple ways to verify authenticity
 - BMFF: e.g. CMAF (HLS, DASH) => **Most common**
 - Hashes: e.g. JPEG, RTP, DSC,... => **What we support**
 - Collections: e.g. group of files
- Why C2PA with DSC instead of C2PA alone?
 - Works with MPEG-TS, RTP
 - Allows remuxing without resigning

Supporting C2PA-DSC in GStreamer

- Wraps c2pa-rs crate
- Only supports C2PA linked to Digitally Signed Content
- Ongoing development ([Draft MR 3197!](#))
 - Supports RSA PKCS#8 Keys
 - Verifies provenance of the video content
 - Generates and verifies C2PA manifests
 - Allow to inject custom compatible C2PA manifests

■ Linking C2PA with DSC in GStreamer



Demo time

Next steps

Next steps

- Merge rust plugins
- H.264
- AI labelling
- Audio

Available resources

- DSC-C2PA WHIP/WHEP demo: <https://github.com/fluendo/c2pa-dsc-demo>
- DSC pipelines: <https://github.com/fluendo/dsc>
- GStreamer issue:
https://gitlab.freedesktop.org/gstreamer/gstreamer/-/work_items/4841
- Fluendo DSC blogpost: <https://fluendo.com/blog/dsc-in-gstreamer/>

Q&A