ISO/IEC JTC 1/SC 29/WG1M97036
96th meeting – Online – October 24-28, 2022

ISO/IEC JTC 1/SC 29/WG 1
(ITU-T SG16)

Coding of Still Pictures

JBIG	**JPEG**
Joint Bi-level Image Experts Group	Joint Photographic Experts Group

TITLE: UPC's answer to the Call for Proposals for JPEG Fake Media: MIPAMS Provenance module

SOURCE: Jaime Delgado, Nikolaos Fotos (DMAG/IMP-UPC)
Distributed Multimedia Applications Group (DMAG) /
Information Modeling and Processing Research Group (IMP)
Universitat Politècnica de Catalunya – UPC BarcelonaTECH
jaime.delgado@upc.edu, nikolaos.fotos@estudiantat.upc.edu

PROJECT: JPEG Fake Media

STATUS: Proposal

REQUESTED ACTION: Input contribution

DISTRIBUTION: WG1

Table of contents

Introduction 3
1. Brief summary 3
2. Detailed technical description 4
  2.1 Data model 6
    2.1.1 Assertion 6
    2.1.2 Assertion Store 11
    2.1.3 Claim 11
    2.1.4 Claim Signature 12
    2.1.5 Credential Store 12
    2.1.6 Manifest 13
    2.1.7 Manifest Store 14
  2.2 Trust model 14
  2.3 Operations 15
    2.3.1 Consume provenance metadata 16
    2.3.2 Produce provenance metadata 17
  3. Implementation and demonstration of the Proposal 19
  3.1 Implementation 19
  3.2 Demonstration of the implementation 22
    3.2.1 Produce provenance metadata 23
    3.2.2 Consume provenance metadata 26
  3.3 MIPAMS architecture 27
4. Requirements compliance table 28
References 30

UPC’s answer to the Call for Proposals for JPEG Fake Media: MIPAMS Provenance module

Introduction

This is the main document of the UPC’s answer to the Call for Proposals for JPEG Fake Media with title “MIPAMS Provenance module”.

The attached document is the Requirements compliance table (Excel file).

Concerning the requested Proposal Elements:

- Brief summary of the proposal: Section 1 of this document.

- Detailed technical description of the proposal: Section 2 of this document.

- Requirements compliance table: Attached document. Section 4 of this documents provides some details on the addressed requirements.

- Implementation for demonstration of the proposal: Section 3 of this document.

1. Brief summary

MIPAMS Provenance module addresses a partial solution for the standardization of “JPEG Fake Media”. It focuses on providing Provenance information through the whole life cycle of the image. In addition, privacy and security issues are considered, trying to keep privacy when needed and protecting the image from manipulation attempts. This is achieved by taking advantage of existing solutions, mainly those in JPEG Systems, such as JUMBF and Privacy and Security.

This proposal covers close to half of the Call requirements, and it should be easy to integrate in other solutions covering the rest of requirements.

The Coalition for Content Provenance and Authenticity (C2PA) [1] published a specification that proposes a universal way to annotate media (e.g., audio files, still images, videos) in order to keep track of its provenance information. In addition, it is unavoidable that there could be provenance information potentially disclosing sensitive information about an entity – person or location. To that extent, C2PA introduces the ability to delete (i.e, redact) the aforementioned information in order to ensure the right to privacy.

The proposed MIPAMS Provenance module complements the C2PA specification by specializing its data model and procedures to work specifically on JPEG images while taking care of the privacy of some provenance information and controlling the access to it.

A complete implementation of the proposal is included, together with a demo application.

The proposed module has been designed and implemented in the context of the MIPAMS architecture, which is used for its implementation and where a new Provenance module is included. This is introduced in subsection 3.3.

2. Detailed technical description

This section presents the formalisation of the specification proposed in this answer to the JPEG Fake Media call for proposals [2]. First and foremost, this specification defines the data model required to express provenance information in a cryptographically secure manner. The data model is based on the work proposed by the technical specification introduced by C2PA [3]. Secondly, this specification provides the means to act upon the provenance information. To that extent, two operations are defined: *Consume* and *Produce*.

This specification adopts the key concepts and notions introduced by C2PA and focuses specifically on expressing provenance information on JPEG images, thus allowing for simplified data structures and operations. Finally, the added value that this specification proposes on top of C2PA architecture is the provision of the use cases that require the protection of a set of metadata that might threaten the privacy of an actor. Alternatively, instead of removing (i.e., redacting) the privacy-sensitive information, this specification adopts the ISO/IEC IS 19566-4:2020 (Privacy and Security) standard [4] in order to protect (i.e., encrypt) sensitive provenance information.

Table 1: Provenance JUMBF Box definitions

Box Name	**Label**	**JUMBF Type (UUID)**	**Explanation**
Manifest Store	mpms. provenance	0x6D707374C65D-11EC-9D64-0242AC120002	The Manifest Store box shall contain at least one Manifest box.
Standard Manifest	[Manifest UUID]	0x6D70736DC65D-11EC-9D64-0242AC120002	The Standard Manifest box shall contain one Claim, one Claim Signature and one Assertion Store and optionally a Credentials Store.

			As for the Assertion Store, a Standard Manifest box shall contain at least one content binding Assertion.
Update Manifest	[Manifest UUID]	0x6D70756DC65D-11EC-9D64-0242AC120002	The Update Manifest box shall contain one Claim, one Claim Signature and one Assertion Store and optionally a Credentials Store. As for the Assertion Store, an Update Manifest box shall contain exactly one Ingredient Assertion that: a) includes a reference to that existing Manifest that is being updated and b) has the value “parentOf” for the relationship field.
Claim	mpms. prov. claim	0x6D70636C0011-0010-8000-00AA00389B71	The Claim box contains only one CBOR Content type box corresponding to the claim structure as described in section 2.1.3.
Claim Signature	mpms. prov. signature	0x6D706373-0011-0010-8000-00AA00389B71	The Claim Signature box contains only one CBOR Content type box to the claim signature structure as described in section 2.1.4.
Assertion Store	mpms. prov. assertions	0x6D706173-C65D-11EC-9D64-0242AC120002	The Assertion Store box contains a list of JUMBF super boxes

			corresponding to the Assertion boxes as specified in section 2.1.2.
Credentials Store	mpms. prov. credentials	0x6D707663-0011-0010-8000-00AA00389B71	The Credentials Store box contains only one JSON Content Type box corresponding to the W3C credential of a user (section 2.1.5).

2.1 Data model

This section presents the proposed data model that is required in order to express provenance information of a JPEG image. The definition of the data model is based on the ISO/IEC IS 19566-5:2022 (JUMBF) standard [5]. This is useful since the provenance information might be embedded inside the related JPEG image. Consequently, new JUMBF Content Types are required in order to express provenance information and to ensure its integrity and authenticity.

Based on the aforementioned requirements, the data model requires the definition of new JUMBF Content Types that will allow the expression of facts regarding a specific digital asset (i.e., image). Each of these facts constitutes an *Assertion*. As explained in more detail in subsection 2.1.1, an Assertion could be expressed using a set of well-known JUMBF Content Types that are already defined in JPEG Systems standards. To ensure that Assertions are stored securely, they should be placed into new JUMBF Boxes. Eventually, one parent JUMBF box, called Manifest Store, should contain all the necessary information to securely express provenance data. A Manifest Store could contain one or more JUMBF Boxes, called Manifests, that describe multiple points in the history of a digital asset.

Furthermore, the specification distinguishes two major entities that act upon provenance information: The Manifest Producers and the Manifest Consumers. The former group consists of the users that produce provenance information (e.g., journalists, artists) while the latter group describes the users that want to validate the authenticity of a digital asset (e.g., social media users).

2.1.1 Assertion

An Assertion is any provenance statement that an actor wants to securely include in the metadata of the JPEG image. An Assertion is represented by a JUMBF Box, whose Content Type can be:

- CBOR [6],
- JSON [7],
- Embedded File, for the case of a thumbnail Assertion, or
- Protection, in case an actor decides to encrypt the content of an Assertion.

The supported Assertions need to cover most of the provenance metadata that an actor might want to embed. These Assertions range from direct modifications of a digital asset, to location metadata and digital rights. Concerning this last concept, it should be noted that the presented approach allows to handle digital rights as assertions. However, how to represent licenses, which should be standardized, is not specified in this proposal.

Table 2 presents a list of the supported Assertion types, originating from different standards like: C2PA, XMP [8], Exif [9].

Table 2: Summary of supported Assertions

Assertion Type	**Assertion Label**	**Origin**
Content Binding	mpms.prov.binding	C2PA
Actions	mpms.prov.actions	C2PA, XMP
Thumbnail	mpms.prov.thumbnail.auto *or* mpms.prov.thumbnail.manual	C2PA
Ingredient	mpms.prov.ingredient	C2PA
Exif	stds.exif	Exif

Subsequently, a graphical design of how an Assertion JUMBF Box should look like is depicted in figure 1.

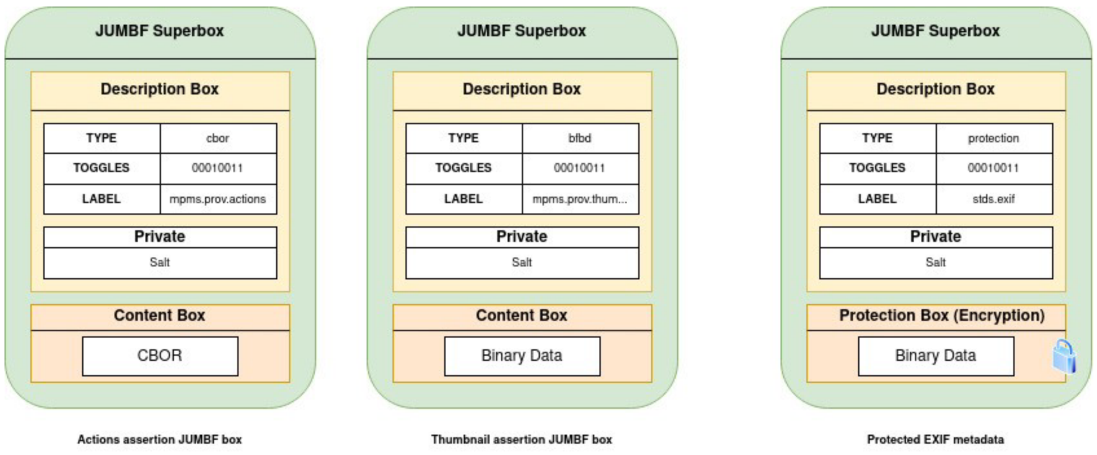


Figure 1: Assertion JUMBF Boxes

As observed in the above figure, TOGGLE is set to 00010011. According to ISO/IEC 19566-5:2022, this means that:

a) a Private Field is present (3rd Most Significant Bit),
b) the JUMBF Box can be requested and
c) the label field exists in the Description box.

Specifically, the label field is a string corresponding to the "Assertion Label" column presented in Table 2. This label shall help the application understand the type of Assertion that the Content Box contains.

Additionally, to add entropy to the hashing process, a Description box's Private field, which acts like a salt, is included. Specifically, a ISOBMFF Box (e.g., Binary Data Box) could be embedded as a Private Field, containing a random string of 16 bytes. As defined in the subsequent subsections, it is important that the hash of the Assertion JUMBF Boxes is calculated in order to verify their integrity. Thus, by using the Private field, it is likely that two Assertions describing an identical modification in a digital asset won't have the same hash digest.

In the case where a producer wants to protect (i.e., encrypt) a subset of the provenance metadata prior to sharing a digital asset, an Assertion could be embedded inside a Protection Content type JUMBF box. Depending on the use case, access rules can be also applied accompanying the Protection Box.

In the following subsections, all the types of Assertions (Content Binding, Actions, Thumbnail, Ingredients, Exif metadata) are described in detail.

2.1.1.1 Content Binding. This Assertion describes the binding between a digital asset and its manifests. Specifically, this Assertion should contain:

- A description of the hash algorithm (e.g., "sha256")
- A padding string (if needed)
- The hash of the digital asset
- A description string (e.g., "Hashing the content of the image by stripping its provenance metadata").

2.1.1.2 Actions. An *actions* Assertion contains a list (e.g., a CBOR list) of the different actions that directly affect the digital asset. Each element of the list shall have a label specifying the type of the action. Table 3 lists all the supported action types.

Table 3: Supported Action Types

Action Type	**Label**	**Origin**	**Explanation**
Convert	mpms.prov.converted	XMP	
Copy	mpms.prov.copied	XMP	
Create	mpms.prov.created	XMP	
Crop	mpms.prov.cropped	XMP	
Edit	mpms.prov.edited	XMP	
Filter	mpms.prov.filtered	XMP	
Format	mpms.prov.formatted	XMP	
Version update	mpms.prov.version updated	XMP	

Print	mpms.prov.printed	XMP	
Publish	mpms.prov.published	XMP	
Manage	mpms.prov.managed	XMP	
Produce	mpms.prov.produced	XMP	
Resize	mpms.prov.resized	XMP	
Save	mpms.prov.saved	XMP	
Place Ingredient	mpms.prov.placed ingredient	C2PA	Added/Placed an ingredient into the asset.
Transcode	mpms.prov.transcoded	C2PA	A direct conversion of one encoding to another, including resolution scaling, bitrate adjustment and encoding format change.
Repackage	mpms.prov.repackaged	C2PA	A conversion of one packaging or container format to another.
Unknown	mpms.prov.unknown	C2PA	Something happened, but the claim generator cannot specify what.

Note: If any of the mp.placed, mp.transcoded, mp.repackaged, mp.unknown actions is used, it is compulsory that there exists a reference to a mp.ingredient Assertion specifying which is the parent digital asset. The contents of an action Assertion shall be serialized with CBOR format and are described below:

- Action: The type of the action as described in "Action Type" column in the table
- Date: the time when the Assertion was produced. It might be different from the date where the claim is produced.
- Metadata: A string providing additional information related to the action.
- Software Agent: A string descriptor related to the application that performed this type of action

2.1.1.3 Thumbnail. A thumbnail Assertion provides an approximate visual representation of the asset at a specific event in the life cycle of an asset. This Assertion is expressed as an Embedded File Content type JUMBF box. The information that needs to be specified for this Assertion is the filename of the thumbnail, the media type and the file itself which shall be embedded in the Provenance Metadata. The label of this Assertion which corresponds to the label of the Description box of the Embedded File Content type JUMBF box - is "mpms.thumbnail".

2.1.1.4 Ingredient. In many scenarios, actors do not create a new asset from scratch; instead, they include other existing assets to create their work - either as a derived asset, a composed asset or an asset rendition (i.e., rendering the digital asset in a different format or quality). These existing assets are called ingredients and they could carry their own provenance history as well. An ingredient Assertion should contain:

- A title for the ingredient. For instance, it could be the name of the ingredient asset.
- The format of the ingredient (Media Type).
- A uniqueID that unambiguously identifies the ingredient digital asset.
- The relationship that the ingredient has with the examined digital asset. The supported relationship types are: “parentOf” and “componentOf”.
- (Optional) A URI reference to the manifest of the ingredient digital asset.
- (Optional) A thumbnail URI Reference to the thumbnail of the ingredient.
- (Optional) Metadata regarding the ingredient itself.

2.1.1.5 Exif metadata. The Exchangeable image file format (Exif) is a common International standard format for storing metadata as part of an image. Exif specification defines a set of tags related to various metadata related to assets (e.g., images, audio files). Exif metadata categories range from image data characteristics (e.g., Color space transformation matrix coefficients) to image data structure (e.g., Artist, Copyright holder) and copyright information.

Exif metadata can be found embedded in a digital asset. However, they can be easily stripped our modified without the permission of the image owner or even without knowing that such a modification has ever taken place. By defining an Assertion type for Exif metadata, it is possible for the provenance producers to specify the changes and ensure that some of this metadata won’t be tampered with.

In addition, Exif metadata might contain information that could raise privacy concerns. For instance, it is quite common that users of social media upload a picture, taken from their mobile phone, containing the GPS location information. Thus, the proposed specification should allow the possibility to redact or control the access of such information to the users that consume the provenance information of the shared digital asset. Exif metadata constitutes the first Assertion type introduced in the specification where both redaction and encryption functionality are supported.

Exif metadata can be modeled as list of key-value pairs (e.g., “Exif.TagName”: “ExampleTagValue”). Consequently, a JSON Content type JUMBF box could contain the Exif metadata that the provenance producer chooses to assert in a provenance claim. In fact, it is highly recommended that the provenance producer application stores the Exif metadata using the JSON-LD schema following the recommendations from XMP. This would allow for a common structure of Exif representation which would result in wider adoption of the proposed specification.

2.1.2 Assertion Store

Assertion Store is a JUMBF box with label “mpms.prov.assertions” and Content Type UUID: 0x6D706173-0011-0010-8000-00AA00389B71. It should contain a list of all the Assertion JUMBF Boxes that shall be included in the claim.

In addition, a Manifest Producer could apply access rules in some Assertions. Hence, each of these Assertion JUMBF Boxes should be replaced by a Protection Box along with a JUMBF Box containing the exact access rules that shall be enforced. For instance, this external JUMBF Box should be an XML Content type JUMBF box containing the access rules using XACML [10].

Note: It is not allowed to encrypt Assertions that describe a major change in the asset. In other words, Protection Box representation is not applicable for Assertions of type Action; it is applicable for Exif metadata Assertion.

2.1.3 Claim

2.1.3.1 Hashed URI Reference

It is essential to define a way to reference different JUMBF Boxes throughout the provenance data structure hierarchy (e.g., when defining Ingredient Assertions).

Thus, the JUMBF reference mechanism specified in ISO/IEC 19566-5:2022 is adopted. Specifically, a URI is defined, consisting of the following parts:

- A prefix stating that the URI is pointing at a JUMBF Box inside the same Manifest Store.
- The manifest unique Id number of the Manifest Content type JUMBF box that the referenced JUMBF Box resides.
- The label of each JUMBF box that wraps the referenced JUMBF box.

An example of such a URI is depicted in Listing 1. In this example, a reference is made to an Action Assertion JUMBF Box located inside the Manifest Content type JUMBF box with label urn:uuid:f977d216-ca0b-4e29-9298-89c8368e5cb9.

```
self#jumbf=mpms/urn:uuid:f977d216-ca0b-4e29-9298-89c8368e5cb9/mpms.assertions/mpms.prov.converted
```

Listing 1: Assertion URI

To ensure integrity - in the sense that the referenced JUMBF Box has not be tampered with after the point of the referencing action - a hash of the JUMBF bytestream needs to be computed. From now on, the set of the URI along with the hash of the referenced JUMBF Box and the hash algorithm used is called *Hashed URI Reference*.

2.1.3.2 Claim definition

A claim constitutes a statement of the following format: “Manifest Producer A has made a set of Assertions for a digital asset 1”. Specifically, a claim contains the following entities:

- a list of URI references of the Assertions residing inside the Assertion Store,
- the URI reference pointing to the Claim Signature (see section 2.1.4),
- list of redacted Assertions (the term “redacted Assertions” refer to the Assertions that a Manifest Producer has decided to delete (i.e., strip) from the Assertion Store for privacy-related reasons): a list of URIs mentioning the label of the Description box of those Assertion JUMBF Boxes that were redacted from the provenance history of the digital asset,
- list of URIs mentioning the label of the Description Box of those Assertion JUMBF Boxes that are protected (i.e., encrypted) inside the Assertion store,
- Claim Generator description: a string that specifies the MIPAMS module along with the specific version where this claim is generated

A claim is represented by a JUMBF box with label “mpms.prov.claim”, Content Type UUID: 0x6D70636C-0011-0010-8000-00AA00389B71 and contains one JUMBF box of type (CBOR).

2.1.4 Claim Signature

Once a Manifest Producer has created a Claim about a particular asset, she needs to digitally sign it and include the digital signature inside another JUMBF box called Claim Signature. This box, a Claim Signature, shall also include the public key that a consumer shall use in order to validate the signature.

Finally, it is of paramount importance to correspond the identity of a producer with the signature that has been generated. This can be achieved through the generation of a digital certificate for each Manifest Producer. This certificate should be generated using the key pair - specifically the private key - that is used by the Manifest Producer to sign the Claim. The generation of such certificates implies the need for establishment of trust anchors around the application.

In scope of RFC 8152 [11], there has been developed a standard way to structure the aforementioned information using CBOR serialization. This scheme is called CBOR Object Signing and Encryption (COSE) and this should be the standard way to express a Claim Signature content. A claim signature is represented by a JUMBF box with label “mpms.prov.signature”, Content Type UUID: 0x6D706373-0011-0010-8000-00AA00389B71 and contains one JUMBF box of type (cbor).

2.1.5 Credential Store

An application might want to correspond a specific actor with the Assertions that are being registered in the provenance history of a digital asset. This doesn’t mean that this actor is necessarily the one that generates the claim or owns the asset. It is a way to provide additional information for the actor that is related to the statements that are being asserted. The way to express this information is through W3C Verifiable Credentials [12] which provide the ability to specify information (i.e., statements/claims) about an actor and can be verified by a provenance consumer.

For this purpose, a new JUMBF Box is introduced. A Credential Store (VCStore) is a JUMBF box that shall contain one or more JSON Content Type boxes (ISO 19566-5:2022, Annex B.4). It shall not contain any other type of JUMBF box or superbox. It shall have a label of "mpms.prov.credentials" and a Content Type UUID: 0x6D707663-0011-00108000-00AA00389B71.

When storing W3C Verifiable Credentials in a VCStore, each one shall be labelled with the value of the id field of the credentialSubject of the VC itself. Since the id is guaranteed to be unique, this ensures that the URI to that credential will be unique.

2.1.6 Manifest

A manifest contains all the information fully describing a single point of the provenance history of an asset. It contains the following information

- Assertion Store: A JUMBF superbox containing all the Assertion JUMBF boxes
- Claim: A JUMBF of CBOR Content type.
- Claim Signature: COSE Structure containing the signature of the actor (Timestamp is also provided)
- Optionally, the Actor's Credentials to bind its identity with the the asset changes metadata

There are two types of Manifest entities: Standard and Update Manifests. Standard Manifest means that the actor is modifying the asset. However, there are cases where an actor doesn't modify the asset but simply wants to include additional metadata (e.g., Exif metadata). In that case an Update Manifest is used. Active Manifest is defined as the latest (i.e., most recent) manifest that describes the current version of an asset.

Since there are two different types of Manifest Entities two different Content types shall be defined. The Standard Manifest shall be a JUMBF box with Content Type UUID: 0x6D70736D-0011-0010-8000-00AA00389B71. Similarly, for the case of Update Manifest, the respective JUMBF box shall have a Content Type UUID: 0x6D70756D-0011-00108000-00AA00389B71. In both cases, the label shall be a uuid uniquely identifying the digital asset. Its format shall be "urn:uuid:<uuid string>".

2.1.7 Manifest Store

Manifest Store is defined as the JUMBF superbox that contains a list of all the Manifest entities of a specific asset. It shall have a label "mpms.prov" and Content Type UUID: 0x6D707374-0011-0010-8000-00AA00389B71.

A Manifest can reference other Manifest JUMBF boxes in the Manifest Store. The final structure of a Manifest Store can be depicted in the two following figure (figure 2):

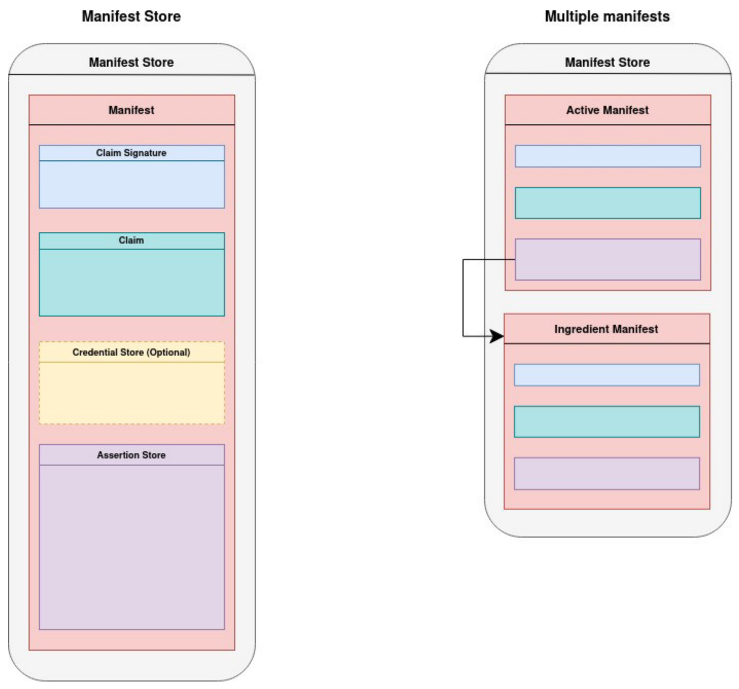


Figure 2: Component hierarchy (From Manifest Store to Assertion)

In the "Multiple manifests" example it is apparent how to express a chain of references describing the entire provenance history of an asset, where older manifests constitute ingredients of the active manifest.

2.2 Trust model

Ensuring trust in all the operations is of outmost importance. In fact, none of the integrity and non-repudiation characteristics of a Manifest would be valid if the identity of the signer is not assured. As specified in section 2.1.4, a digital certificate needs to be issued for a Manifest Producer in order to be able to generate provenance information. By digitally signing the claim using the same private key related to the certificate, a Manifest Consumer can validate the identity of the actor who generates the Assertions. This implies that as part of the operations described in subsection 2.3, it is imperative to include the validity of the embedded certificates.

Thus, for any provenance application that follows this specification it is important to define the entities that are responsible for controlling/authorizing the ability to produce provenance information. These entities are called trust anchors. In this case, the trust anchors can be conceived as the Certification Authorities (CA) that provide the means to generate certificates for provenance producers as well as the interface which allows applications to query and validate the correctness of a certificate.

As clearly stated in C2PA technical specification, all the trust signals have as final destination the "Signer" which is the provenance producer. This trust relationship is depicted in figure 3.

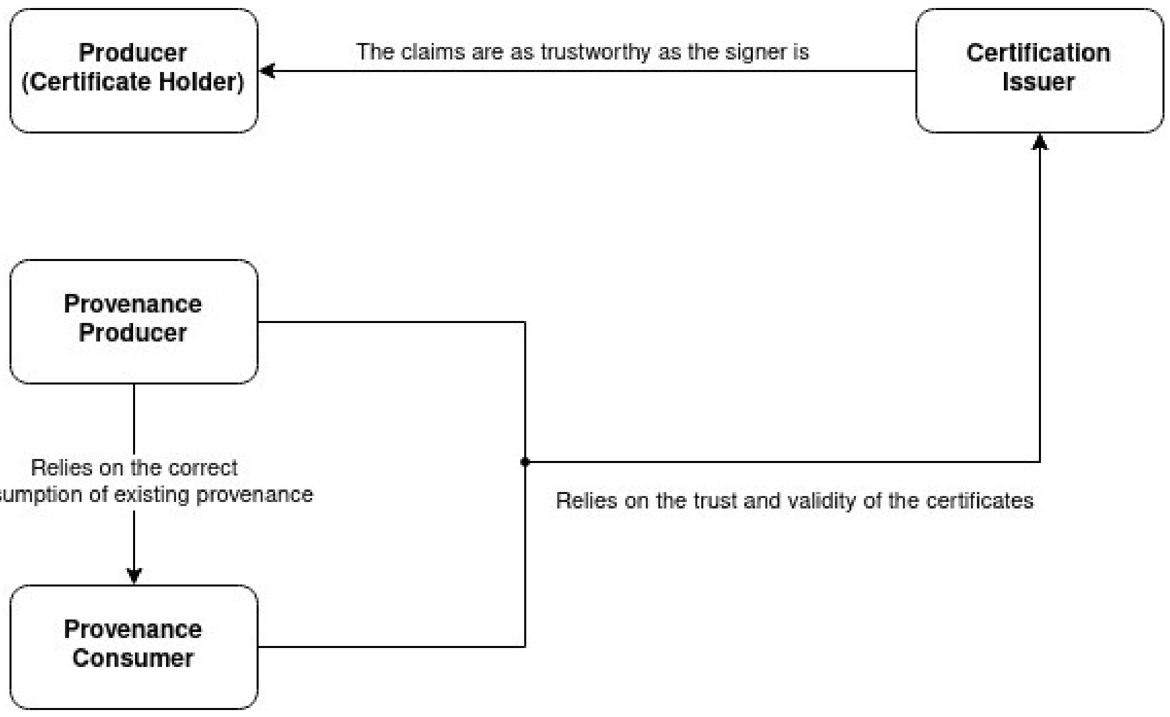

Producer (Certificate Holder)
The claims are as trustworthy as the signer is
Certification Issuer
Provenance Producer
Relies on the correct consumption of existing provenance
Relies on the trust and validity of the certificates
Provenance Consumer

Figure 3: Trust relationship between provenance actors and entities

A simplified example of such a trust model could be applied in the case of journalism. A set of big journalist organizations around the world could play the role of trust anchors. All the international and national news agencies should be registered in at least one of them. A relationship of trust is established between them. News agencies could play the role of intermediate CAs that are authorized by the root CA to issue certificates for their entities. Eventually, this builds an entire chain of trust at the bottom of which there are the journalists.

To sum up, with these certifications chains it is possible to correspond the provenance claims with specific actors. It is assumed that the validation of the producers' certificates is handled by the application that uses the implementation of this standard.

2.3 Operations

This section describes the logic of the two core operations that a user can perform with provenance metadata. Specifically, two algorithms are presented: one related to consuming and one to producing provenance metadata. It is worth mentioning that in both algorithms the provenance information is provided separately to the digital asset. The goal is to support use cases and applications that embed provenance information into the digital asset as well as those that store this metadata outside of the digital asset. Regarding the first case, the specification assumes that the application shall handle the extraction and stripping of the provenance metadata from the digital asset before using the following algorithms.

2.3.1 Consume provenance metadata

The first operation defines how to consume provenance metadata related to a digital asset. Provenance metadata is stored in the form of a Manifest Store Content type JUMBF box as defined in section 2.1. Depending on the use case the consumer application might select to consume the entire provenance history that is stored in the asset (i.e., all the Manifest Content type JUMBF boxes inside the Manifest Store) or simply the Active Manifest Content type JUMBF box assuming that the rest of the provenance chain is already validated. In general, validating a Manifest consists of validating its Claim Signature as well as the integrity of the Assertions through the Assertion references (i.e., Hashed URI References) located in the Claim. In the case of a Standard Manifest, validation includes also the validation of the content binding Assertion that verifies the connection between the Manifest and the digital asset. As explained in section 2.1.1.1, in the case where the Active Manifest is an Update Manifest then the content binding Assertion shall be retrieved from the first parent Standard Manifest.

Throughout this operation, the term "digital asset" refers to the image codestream without the Manifest Content type JUMBF box. It is of paramount importance that every time the digest of an asset is calculated, the provenance metadata bytestream is not taken into consideration. Consequently, the application that uses this algorithm shall strip the provenance metadata beforehand. With this approach it is possible to avoid the circular dependency where the digest of the asset assumes that the provenance information is finalized and embedded into the asset. However, for this to happen, the digest must have been already computed.

Let's consider the case when a Manifest Consumer wants to perform a fully inspection of a Manifest Store that contains Ingredient Manifests along with an Active Manifest. All manifests need to be inspected and the Manifest Consume algorithm is performed recursively until all the provenance chain is validated. The algorithm is depicted in algorithm 1.

ALGORITHM 1: ALGORITHM TO CONSUME PROVENANCE METADATA

```
1   procedure INSPECT-PROVENANCE(manifest_store, asset, is_full_inspection)
2       active_manifest ← locateActiveManifest(manifest_store)
3       is_standard ← isStandardManifest(active_manifest)
4       if is_standard then
5           verifyManifestIntegrity(active_manifest)
6           verifyContentBinding(active_manifest, asset)
7       else
8           while not is_standard do
9               manifest ← locateParentManifest(active_manifest, manifest_store)
10              is_standard ← isStandardManifest(manifest)
11              verifyManifestIntegrity(manifest)
12          verifyContentBinding(manifest, asset)
13      to_be_checked ← [ ]
```

```
    if is_full_inspection then
        to_be_checked $\leftarrow$ addComponentIngredients(active_manifest)
        while isEmpty(to_be_checked) do
            manifest_reference $\leftarrow$ to_be_checked.next()
            manifest $\leftarrow$ locateManifest(manifest_reference, manifest_store)
            verifyManifestIntegrity(manifest)
            to_be_checked $\leftarrow$ addComponentIngredients(manifest)
    result $\leftarrow$ active_manifest
    return result
procedure VERIFY-MANIFEST-INTEGRITY(manifest)
    validateClaimSignature(manifest)
    validationAssertionIntegrity(manifest)
```

2.3.2 Produce provenance metadata

Producing provenance metadata corresponds to any addition of provenance point in the history of a digital asset. This means that there are multiple use cases that need to be taken into consideration. First, the algorithm starts by providing the current state of an asset along with its provenance history that has been recorded so far. On top of that, the producer application that calls the algorithm, needs to provide the asset after the new modifications have taken place as well as the Assertions that describe them. These Assertions could be of any type out of the ones defined in section 2.1.1. The Assertions are provided as a list of JUMBF Boxes ready to be embedded in the Assertion Store. This allows the producer applications to perform any encryption and apply any access rules policy they fit proper for their use case. Regarding the redaction process, the producer application could provide a set of URI References, pointing to the Assertions that shall be redacted. These references point to Assertions located in ingredient manifests inside the asset's Manifest Store. Not all Assertion types can be redacted. Finally, in case the producer application wants to provide Verifiable Credentials corresponding the Assertions to an entity, the Credential Store Content type JUMBF box must be provided to the algorithm in advance.

Now that all the possible input parameters have been described, the algorithm starts by checking whether there is provenance history already registered to the digital asset. In this case, before the new manifest is produced, it is crucial that at least the current active manifest of the digital asset is verified to ensure that the new Manifest is not added on top of a corrupted or malformed provenance chain. In addition, it is also important to verify that the Assertions that compose the new claim contain the proper parent ingredient Assertion referencing (i.e., Hashed URI reference) the current active manifest.

Next, the new Manifest shall be produced. Depending on the type of the Manifest (i.e., depending on the Assertions provided), a content binding Assertion might be needed in order to relate the new Manifest to the revised digital asset. Since all the Assertions are provided in the form of JUMBF Boxes, they need to be wrapped with an Assertion Store Content type JUMBF box. Then the Claim Content type JUMBF box is constructed consisting of all the fields described in section 2.1.3. Now, the producer application can digitally sign the constructed Claim using the private key that is used to generate the digital certificate corresponding to the producer actor who builds this provenance point. All these boxes - along with, optionally, the Credentials Store box are wrapped inside a new Manifest Content type JUMBF box which constitutes now the latest provenance point in the digital asset's provenance chain.

Moreover, the algorithm needs to take into consideration the use cases where the producer includes a set of ingredients assets that have their own provenance history. In that case, the active manifest of these ingredient manifest stores is validated and included in the digital asset's manifest store.

Finally, now that the manifest store is updated, the algorithm needs to handle the "Redact Assertions" request. This request is related to the "redactable" Assertions of the ingredient manifests that reside inside the Manifest Store Content type JUMBF box. The redacted Assertions, that are already included in the new active manifest's claim, need to be removed from the ingredient Assertions of the entire provenance history of the digital asset. The Manifest Store is, eventually, complete and the summarized algorithm of producing it is depicted in algorithm 2.

ALGORITHM 2: ALGORITHM TO PRODUCE PROVENANCE METADATA

```
procedure PROVENANCE-PROVENANCE(CA, NA, MS, AL, RL, IM, signer, CS)
    // CA: Current Asset, NA: New Asset
    // MS: Manifest Store, AL: Assertion List
    // RL: Redaction List, IM: Ingredient Manifest List,
    // CS: Credentials Store
    if MS exists then
        PROVENANCE-CONSUME(CA, False)
        active_manifest ← locateActiveManifest(MS)
        checkParentIngredientAssertion(AL, active_manifest)
    else
        MS ← createEmptyManifestStore()
    new_active_manifest ← PRODUCE-MANIFEST(NA, AL, RL, signer, CS)
    if IM exists then
        validateUriReferences(AL, IM)
        MS ← MS + IM
    if RL exists then
        manifest_reference ← to_be_checked.next()
    MS ← MS + active_manifest
    Return MS
procedure PRODUCE-MANIFEST-INTEGRITY(asset, AL, RL, signer, CS)
    manifest_id ← issueNewUuid()
    Manifest_type ← discoverManifestType(AL)
    if manifest_type is 'Standard Manifest' then
        ABox ← buildAssertionStoreWithContentBindingAssertion(asset, AL)
    else
```

```
        ABox ← buildAssertionStore(AL)
    CBox ← buildClaim(manifest_id, ABox, RL)
    CSBox ← buildClaimSignature(signer, CBox)
    manifest ← buildManifest(ABox, CBox, CSBox, CS)
    return manifest
```

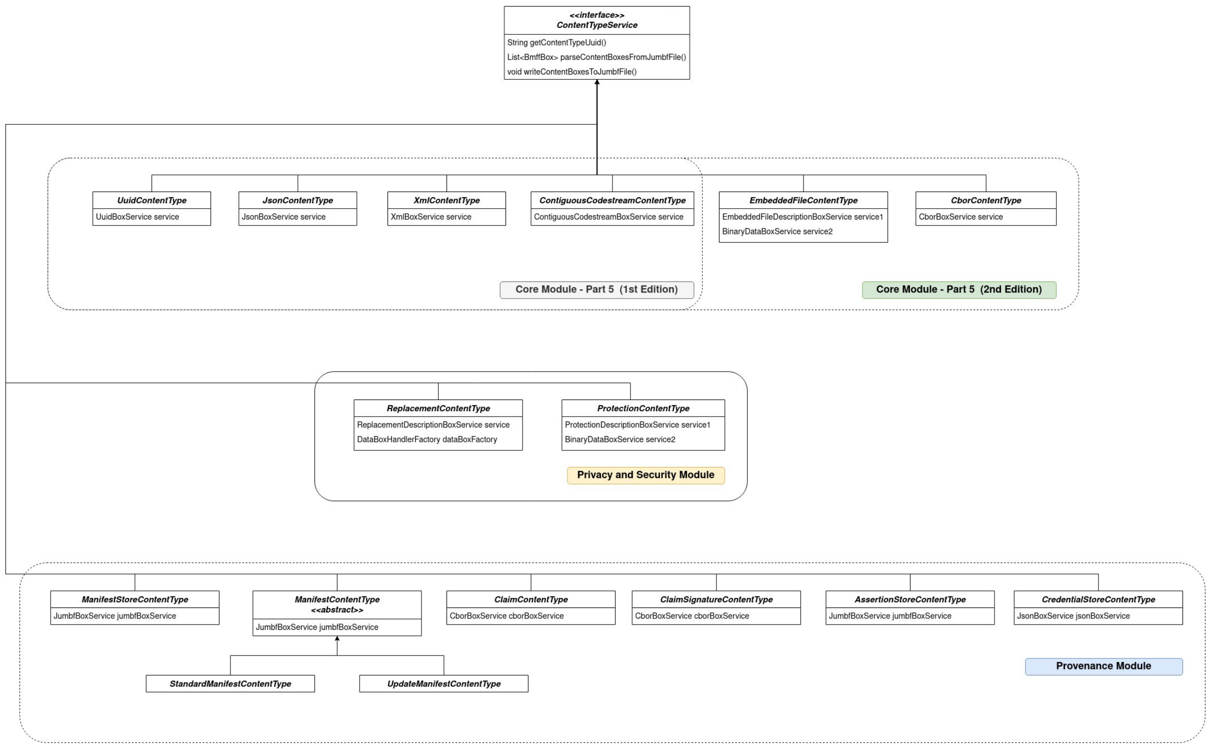
3. Implementation and demonstration of the Proposal

3.1 Implementation

This section presents an implementation of the proposed specification. The library, called mipams-fake-media, implements the proposed provenance data model as well as the operations to produce and consume such data. The source code is available at the following link: https://github.com/DMAG-UPC/mipams-fake-media

This library is based on an existing project that implements the JUMBF data model and the functionalities to handle (i.e., parse/generate) such structures. This project is called mipams-jumbf and it is an open-source Java-based project that consists of a collection of libraries that support various JUMBF box definitions from various JPEG Systems standards. Currently, the following JPEG Systems standards are supported: ISO/IEC 19566-5:2022 (JUMBF), ISO/IEC 19566-4:2020 (Privacy & Security) and ISO/IEC 19566-7:2022 (JLINK). For the provenance use case, the JUMBF Boxes from the first two standards are used. The mipams-jumbf libraries that are used, namely the jumbf-core (JUMBF) and jumbf-privsec (Privacy & Security) are also proposed as reference software for the respective JPEG Systems standards [13], [14].

Based on the definitions proposed in subsection2.1, all the JUMBF Box structures (i.e., Description and Content Boxes) are already implemented in jumbf-core-2.0 and jumbf-privsec-1.0 libraries. Consequently, what needs to be implemented for the provenance data model is the Content Type classes as illustrated in the "Provenance Module" in figure 4.



<<interface>>
ContentTypeService
String getContentTypeUuid()
List<BmffBox> parseContentBoxesFromJumbfFile()
void writeContentBoxesToJumbfFile()
UuidContentType
UuidBoxService service
JsonContentType
JsonBoxService service
XmlContentType
XmlBoxService service
ContiguousCodestreamContentType
ContiguousCodestreamBoxService service
EmbeddedFileContentType
EmbeddedFileDescriptionBoxService service1
BinaryDataBoxService service2
CborContentType
CborBoxService service
Core Module - Part 5 (1st Edition)
Core Module - Part 5 (2nd Edition)
ReplacementContentType
ReplacementDescriptionBoxService service
DataBoxHandlerFactory dataBoxFactory
ProtectionContentType
ProtectionDescriptionBoxService service1
BinaryDataBoxService service2
Privacy and Security Module
ManifestStoreContentType
JumbfBoxService jumbfBoxService
ManifestContentType
<<abstract>>
JumbfBoxService jumbfBoxService
ClaimContentType
CborBoxService cborBoxService
ClaimSignatureContentType
CborBoxService cborBoxService
AssertionStoreContentType
JumbfBoxService jumbfBoxService
CredentialStoreContentType
JsonBoxService jsonBoxService
StandardManifestContentType
UpdateManifestContentType
Provenance Module

Figure 4: Extended Content Type class hierarchy containing the Provenance module

At the bottom level of figure 4, the Provenance Content Types are depicted. It's worth noting that each Content Type uses a Box service that is already defined in the jumbf-core-2.0. Specifically, in the cases of ClaimContentType, ClaimSignatureContentType and CredentialStoreContentType classes, only one specific box service is required. However, this is not the case for the remaining Provenance Boxes. As explained in subsection 2.1.2, the content of an Assertion Store Content type JUMBF box is a list of JUMBF Boxes that is of type CBOR, JSON, Embedded File or Protection. This is the reason why a JumbfBoxService field is assigned for the AssertionStoreContentType class. Regarding the Manifest Content type JUMBF box, it contains a set of JUMBF Boxes (i.e., Claim, Claim Signature, Assertion Store and – optionally – one Credential Store). Thus, the only Box Service that is required is the JumbfBoxService. Finally, the same applies for a Manifest Store Box which contain a list of Manifest Content type JUMBF boxes.

In addition to the new JUMBF Box definitions, the provenance operations specified in 2.3 are implemented as well. Different service classes are implemented for each part of the "Consume" and "Produce" provenance operations. As in the case of Content Type classes, they are implemented as Java Beans facilitating the dependency injection for each of these classes. The dependency graph of the service classes for each of the two provenance operations is depicted in figures 5 and 6.

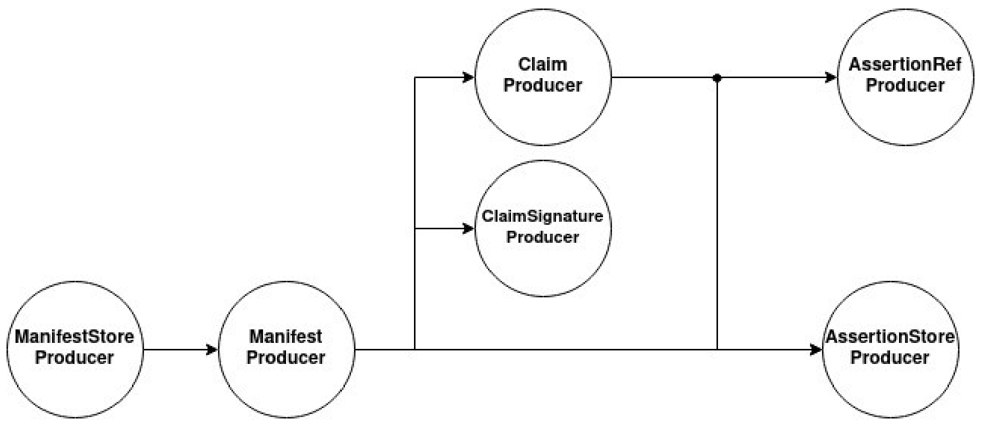


Figure 5: Producer services dependency graph

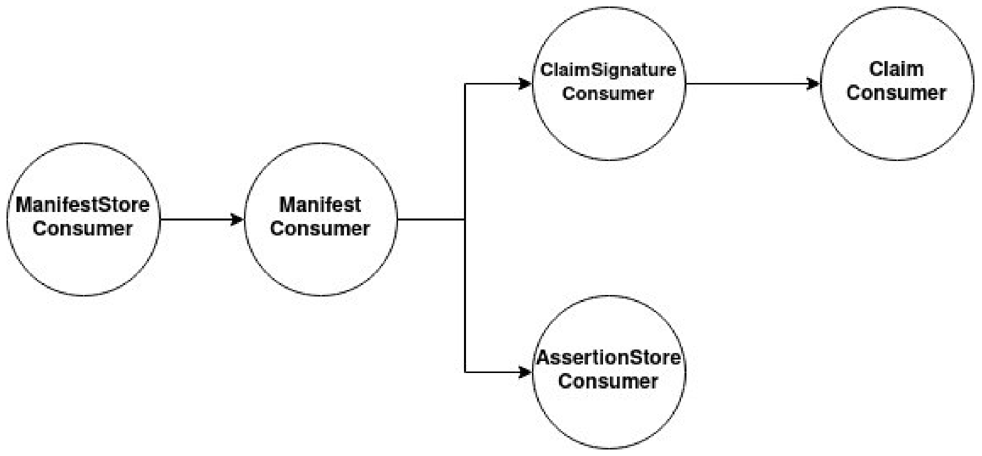


Figure 6: Consumer services dependency graph

Both “Consume” and “Produce” provenance operations require a set of cryptographic tools in order to handle digital signatures, encrypted Assertions etc. As a result, a new library called “jumbf-crypto” is implemented inside the mipams-jumbf project and performs a set of cryptographic operations (i.e., encrypt, decrypt, sign, validate signature, parse keys from files). These cryptographic operations are implemented using the Java Cryptography Architecture (JCA) [15]. Currently in jumbf-crypto-1.0, the following cryptographic operations are supported:

- Generating and validating digital signatures: SHA1 with RSA
- Encrypting/Decrypting a byte stream: AES-256
- Calculating the Hash of a byte stream: SHA256

Finally, in mipams-fake-media library the set of Assertions is specified according to section 2.1. Although the provenance operations are implemented handling the Assertions as JUMBF Boxes, it is imperative that this library defines the structure of the content of each Assertion JUMBF Box. Moreover, all the serialization and deserialization operations (i.e., extracting information from the JUMBF Box Content Boxes or storing Assertion information into a

JUMBF Box) are implemented as well. The class hierarchy of the Assertion classes is depicted in figure 7. Notice that there are two interfaces, namely RedactableAssertion and NonRedactableAssertion that assist with the definition of each Assertion class. As of mipams-fake-media-1.0 only ExifMetadata Assertion is redactable.

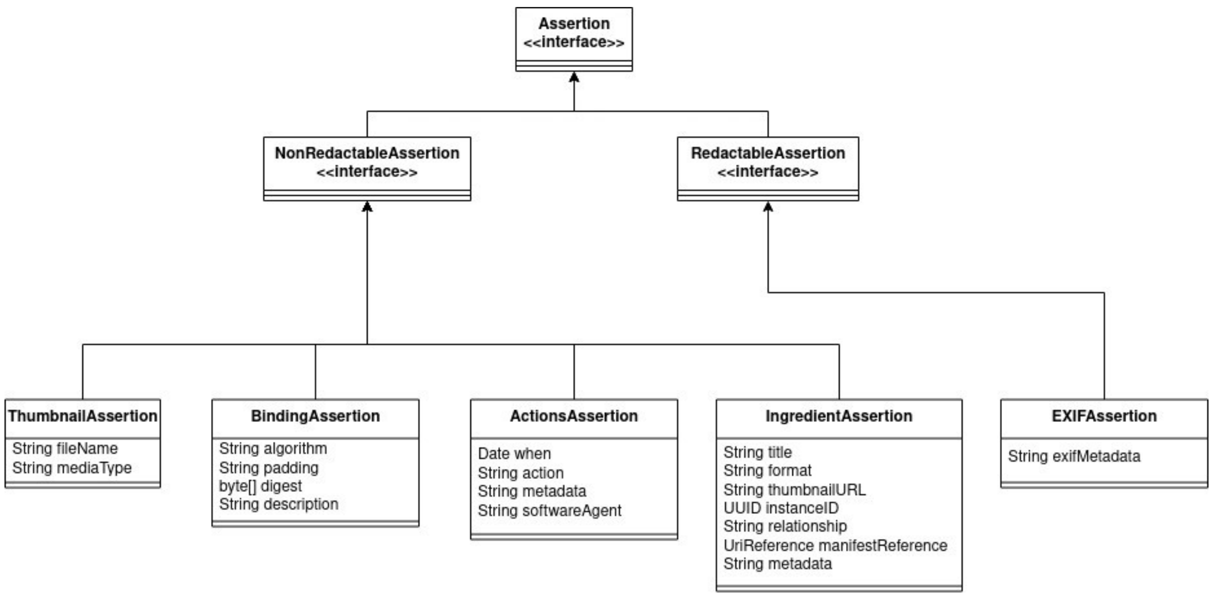


Figure 7: Assertions class hierarchy

3.2 Demonstration of the implementation

This section illustrates an example application that allows authenticated users (i.e., producers) to create provenance information for a specific digital asset. This provenance information is embedded as a Manifest Store inside the digital asset, which can be then shared across multiple users that will try to view it (i.e., consume). The goal of this application is to focus on the use cases where a producer wants to protect/control the access to a set of metadata that could disclose personal information, such as GPS Location. For this reason, a set of roles have been defined for the authenticated users of this application: Consumers and Producers.

This demonstration simulates an application related to the journalism field where producers are members of journalist associations. The examined workflow consists of a journalist who selects to protect the Exif metadata (i.e., contain GPS and device specific information) of a JPEG image. In addition, the journalist wants to apply access rules in order to control the access to this piece of metadata so that only fellow journalists have access to this metadata. Upon registration to the application, journalists (i.e., producers) have received a key pair and a digital certificate. As explained before, the digital certificate is crucial for the producer in order to generate - i.e., digitally sign - provenance information.

The provenance metadata manager application is developed using Java and Spring boot framework for the RESTfull API, while the front-end application is written using React JS. The entire provenance demonstrator application is located at the following URL: https://github.com/DMAG-UPC/mipams-fake-media-demonstrator.

3.2.1 Produce provenance metadata

In figure 8, it is assumed that a journalist has logged in to the provenance metadata manager and has uploaded an image. At this stage, the journalist has the ability to apply some direct modifications to the digital asset. All modifications are recorded and placed as Assertions of type "actions".

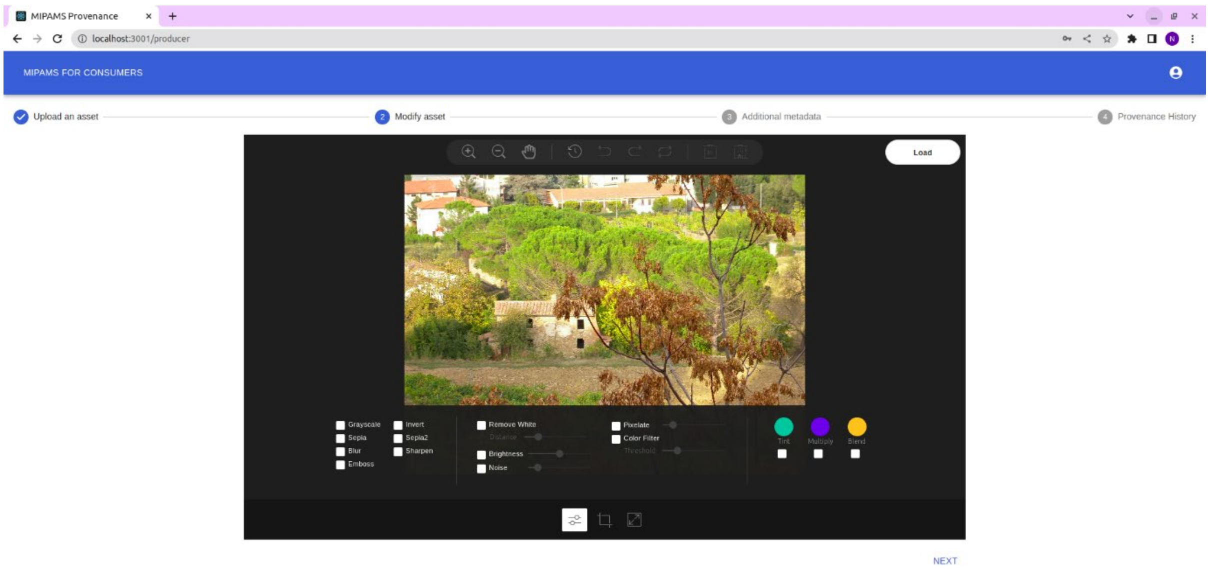


Figure 8: A digital asset loaded in the provenance manager application

For this example, the journalist performs the following changes to the image: Filter the image (adding Sepia filter, blurring and reducing the brightness), crop the image and change its size. The resulted image is depicted in figure 9.

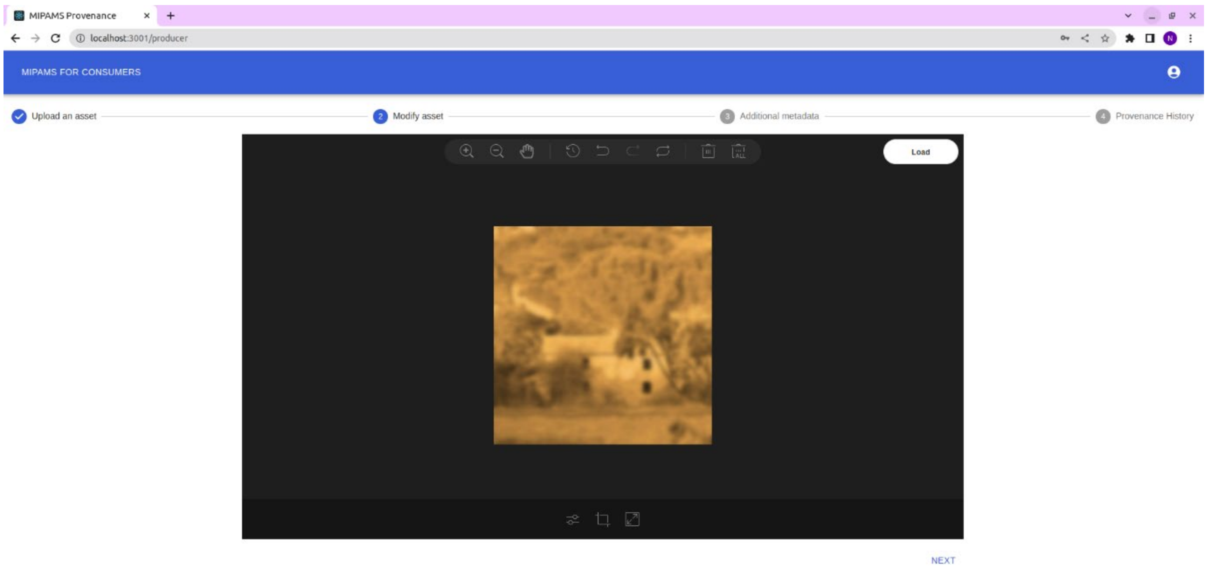
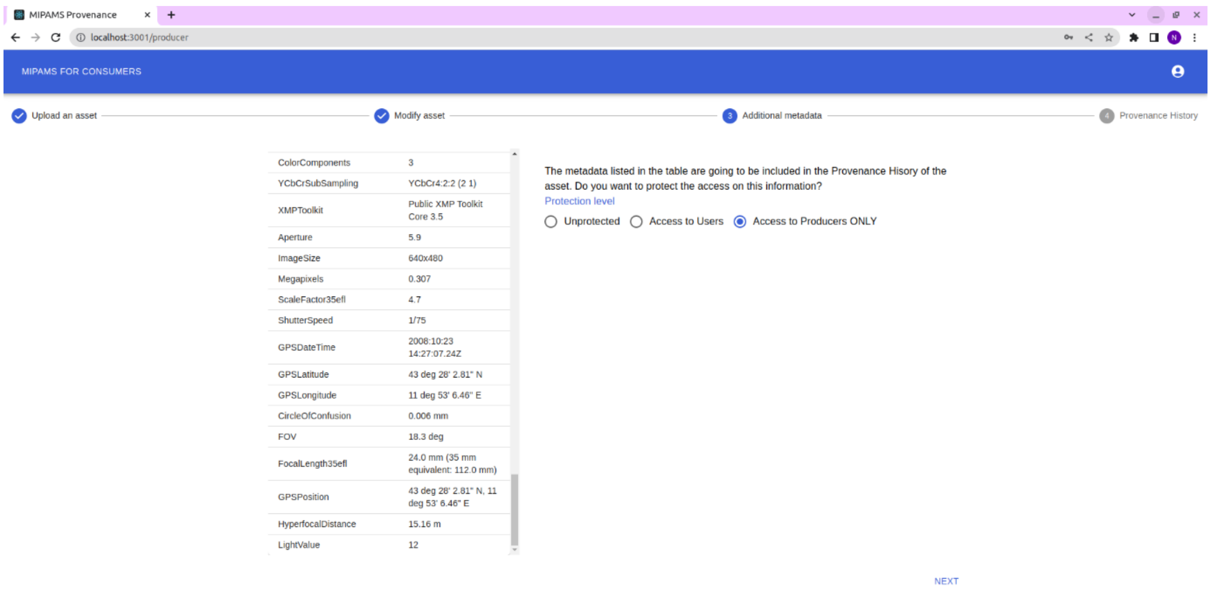


Figure 9: The uploaded digital asset upon modifications

Next, the journalist can view the Exif metadata that are embedded inside the digital asset. As shown in figure 10, this metadata contains information that she might select not to disclose to all the users who might view this asset. As depicted in the figure below, the producer has three options: (i) include EXIF metadata unprotected in the Assertion Store Content type JUMBF box, (ii) encrypt them as a Protection box that shall be accessible only by authenticated users of this application or (iii) apply access rules so that only "Producer" users (i.e., journalist) have access to this metadata.



ColorComponents	3
YCbCrSubSampling	YCbCr4:2:2 (2 1)
XMPToolkit	Public XMP Toolkit Core 3.5
Aperture	5.9
ImageSize	640x480
Megapixels	0.307
ScaleFactor35efl	4.7
ShutterSpeed	1/75
GPSDateTime	2008:10:23 14:27:07.24Z
GPSLatitude	43 deg 28' 2.81" N
GPSLongitude	11 deg 53' 6.46" E
CircleOfConfusion	0.006 mm
FOV	18.3 deg
FocalLength35efl	24.0 mm (35 mm equivalent: 112.0 mm)
GPSPosition	43 deg 28' 2.81" N, 11 deg 53' 6.46" E
HyperfocalDistance	15.16 m
LightValue	12

The metadata listed in the table are going to be included in the Provenance Hisory of the asset. Do you want to protect the access on this information?

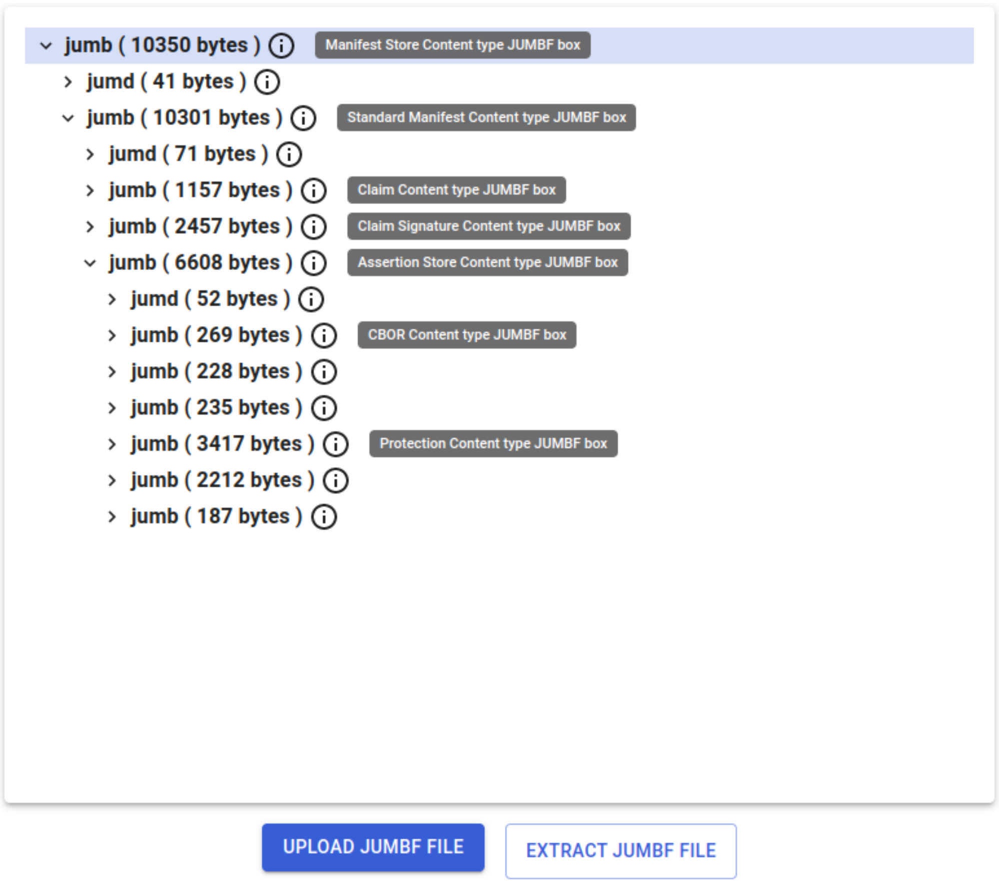
Protection level

Unprotected Access to Users [✓ Access to Producers ONLY]

NEXT

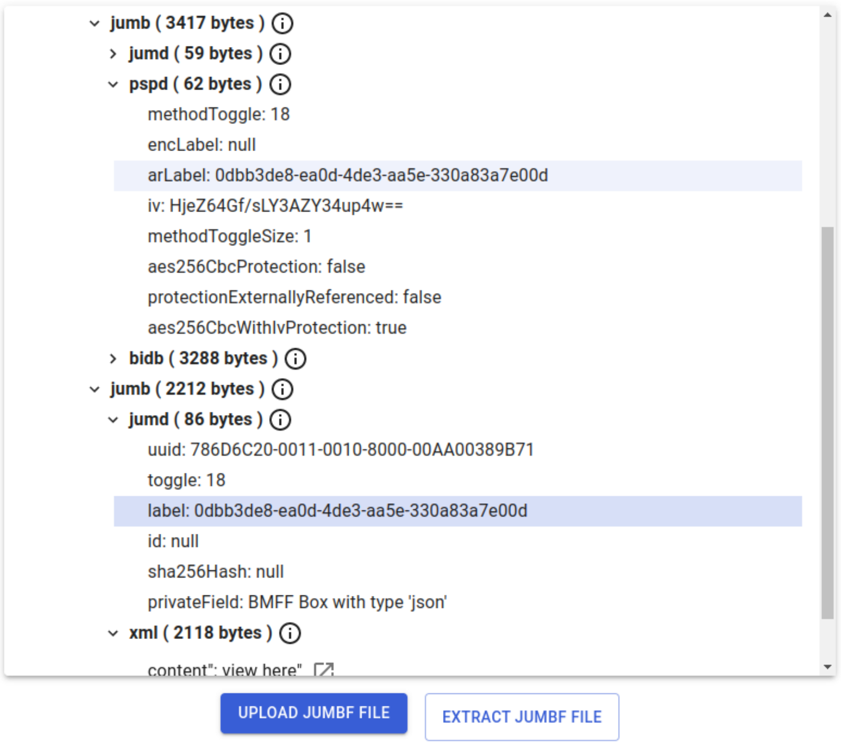
Figure 10: The uploaded digital asset upon modifications

The producer selects the third option and proceeds with the generation of the provenance metadata. As a result, a new JPEG image is generated with the Manifest Store Content type JUMBF box embedded inside. To inspect the data structure of the embedded JUMBF box, the demo application of the mipams-jumbf project is used. In figure 11, the Manifest Store Content type JUMBF box is depicted, including all the boxes as defined in subsection 2.1. The structure could be further investigated; in figure 12 it is evident that one of the Assertions JUMBF Boxes is a Protection Content type JUMBF box since the journalist selected to encrypt (i.e., protect) the Exif metadata information. Inside the Protection Description box there is also a reference to a JUMBF box that contains access rules. The latter box is an XML Content type JUMBF box that contains the access rules expressed in XACML (figure 13).



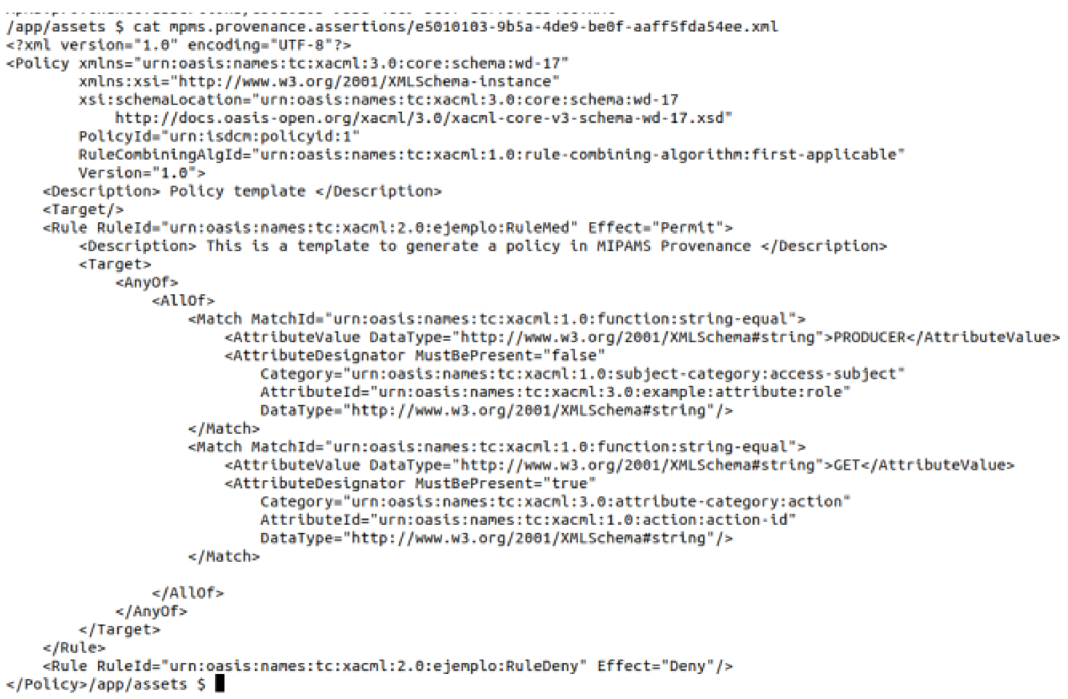
```
jumb ( 10350 bytes )   Manifest Store Content type JUMBF box
  jumd ( 41 bytes )
  jumb ( 10301 bytes )   Standard Manifest Content type JUMBF box
    jumd ( 71 bytes )
    jumb ( 1157 bytes )   Claim Content type JUMBF box
    jumb ( 2457 bytes )   Claim Signature Content type JUMBF box
    jumb ( 6608 bytes )   Assertion Store Content type JUMBF box
      jumd ( 52 bytes )
      jumb ( 269 bytes )   CBOR Content type JUMBF box
      jumb ( 228 bytes )
      jumb ( 235 bytes )
      jumb ( 3417 bytes )   Protection Content type JUMBF box
      jumb ( 2212 bytes )
      jumb ( 187 bytes )
```

Figure 11: Inspection of provenance JUMBF structure in generated digital asset



```
jumb ( 3417 bytes )
  jumd ( 59 bytes )
  pspd ( 62 bytes )
    methodToggle: 18
    encLabel: null
    arLabel: 0dbb3de8-ea0d-4de3-aa5e-330a83a7e00d
    iv: HjeZ64Gf/sLY3AZY34up4w==
    methodToggleSize: 1
    aes256CbcProtection: false
    protectionExternallyReferenced: false
    aes256CbcWithIvProtection: true
  bidb ( 3288 bytes )
jumb ( 2212 bytes )
  jumd ( 86 bytes )
    uuid: 786D6C20-0011-0010-8000-00AA00389B71
    toggle: 18
    label: 0dbb3de8-ea0d-4de3-aa5e-330a83a7e00d
    id: null
    sha256Hash: null
    privateField: BMFF Box with type 'json'
  xml ( 2118 bytes )
    content": view here"
```

Figure 12: Protected Assertion with access rules embedded in an XML Content type JUMBF box



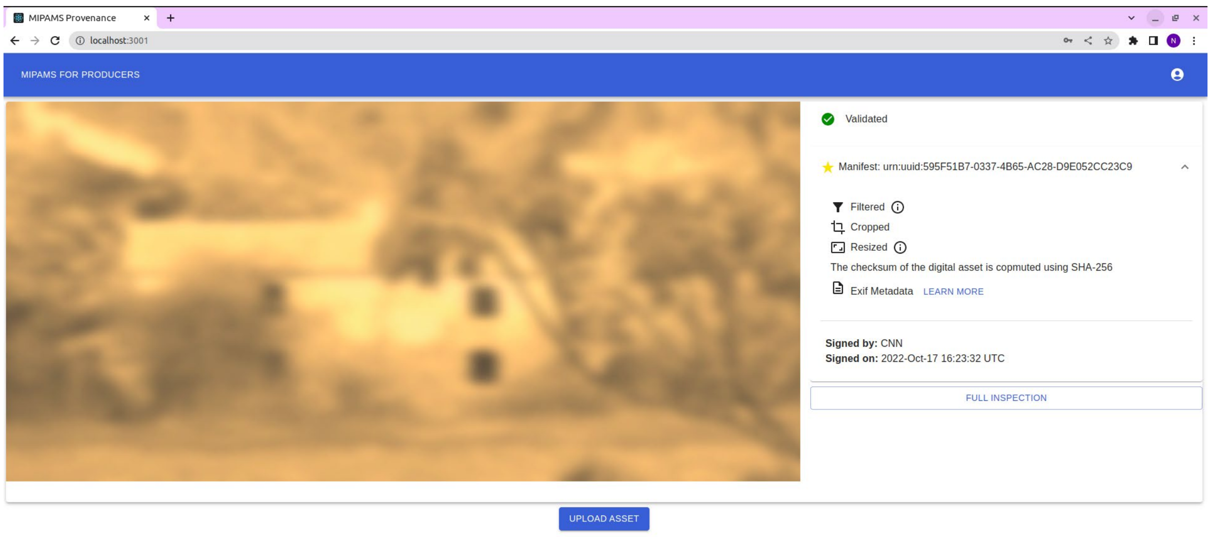
```
/app/assets $ cat mpms.provenance.assertions/e5010103-9b5a-4de9-be0f-aaff5fda54ee.xml
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17
            http://docs.oasis-open.org/xacml/3.0/xacml-core-v3-schema-wd-17.xsd"
        PolicyId="urn:isdcm:policyid:1"
        RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable"
        Version="1.0">
    <Description> Policy template </Description>
    <Target/>
    <Rule RuleId="urn:oasis:names:tc:xacml:2.0:ejemplo:RuleMed" Effect="Permit">
        <Description> This is a template to generate a policy in MIPAMS Provenance </Description>
        <Target>
            <AnyOf>
                <AllOf>
                    <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">PRODUCER</AttributeValue>
                        <AttributeDesignator MustBePresent="false"
                            Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
                            AttributeId="urn:oasis:names:tc:xacml:3.0:example:attribute:role"
                            DataType="http://www.w3.org/2001/XMLSchema#string"/>
                    </Match>
                    <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">GET</AttributeValue>
                        <AttributeDesignator MustBePresent="true"
                            Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
                            AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
                            DataType="http://www.w3.org/2001/XMLSchema#string"/>
                    </Match>

                </AllOf>
            </AnyOf>
        </Target>
    </Rule>
    <Rule RuleId="urn:oasis:names:tc:xacml:2.0:ejemplo:RuleDeny" Effect="Deny"/>
</Policy>/app/assets $ █
```

Figure 13: XACML policy: Only users with Role "PRODUCER" can access the resource

3.2.2 Consume provenance metadata

At this point of the example, the digital asset is shared across social networks and it is assumed that another journalist downloads the asset, logs in to the provenance metadata manager and uploads the asset. As shown in figure 14, there is a right-side bar that contains provenance information. First and foremost, the application notifies the user that provenance information has been validated successfully. Next, the manifest Id and the Assertions are depicted in detail. It is worth mentioning that the journalist has access to the the content of the Exif metadata and by clicking on the "Learn More" button, the journalist can inspect the Exif metadata successfully. Finally, information about the signer of the provenance information is depicted.



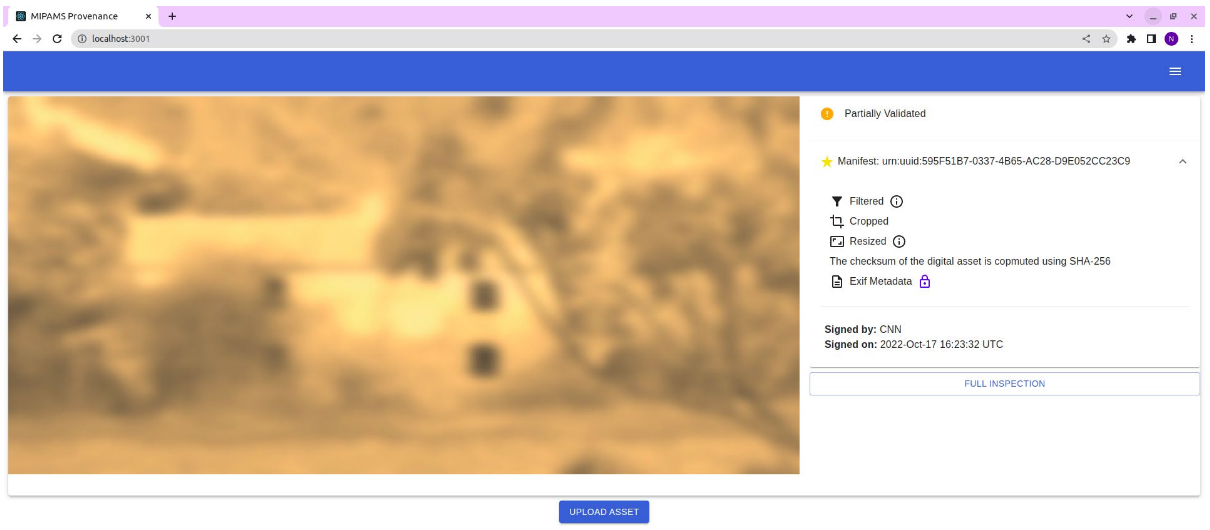
Manifest: urn:uuid:595F51B7-0337-4B65-AC28-D9E052CC23C9

Filtered
Cropped
Resized
The checksum of the digital asset is copmuted using SHA-256
Exif Metadata LEARN MORE

Signed by: CNN
Signed on: 2022-Oct-17 16:23:32 UTC

Figure 14: Producer consumes provenance information of a digital asset

An example of a user that doesn't have access to the Exif metadata of the digital asset is depicted in figure 15. The status of the provenance information inspection is set to "Partially Validated" since the user doesn't have access to the Exif metadata and a lock icon is also depicted.



Partially Validated

Manifest: urn:uuid:595F51B7-0337-4B65-AC28-D9E052CC23C9

Filtered

Cropped

Resized

The checksum of the digital asset is copmuted using SHA-256

Exif Metadata

Signed by: CNN
Signed on: 2022-Oct-17 16:23:32 UTC

Figure 15: Unauthorized user cannot view the proteted metadata

3.3 MIPAMS architecture

The demonstrator just presented in this subsection is a proof-of-concept application explaining the provenance operations; it does not specify the way to manage the producers' digital certificates used for signing the claims nor does it provide an external service which creates the XACML policies and requests for the inclusion of privacy rules that protect the Exif metadata of a digital asset. These aforementioned required services are already implemented in the Multimedia Information Protection And Management System (MIPAMS) [16]. This section focuses on defining the interconnections that need to be implemented in order for a Provenance application to operate inside the MIPAMS environment. The architecture of the MIPAMS environment integrating a service that handles provenance information is depicted in figure 16.

The Provenance module needs to communicate with multiple modules of the MIPAMS architecture in order to perform the operations defined in subsection 2.3. First, the creation and storage of users' credentials (e.g., keys, certificates) is handled by the MIPAMS Protection Service, a module that is only accessible from authenticated and authorized services.

In addition, all the cryptographic operations described so far need to be implemented in a specific service that is authorized to access the specific producer private key and certificate. Moreover, this service should properly handle, as well as destroy, these credentials to avoid any leakage of cryptographic material that would compromise the entire security of the solution. In MIPAMS environment, this module is called Content service.

Regarding the generation and validation of privacy rules, the MIPAMS environment defines a Policy Service (PoS) that is responsible for creating a XACML policy according to a set of provided parameters. In parallel, this service is responsible for creating a XACML request according to the authenticated user information. With these two pieces of information, the Provenance module could validate whether a user has the correct access rules by communicating with the MIPAMS Authorization service.

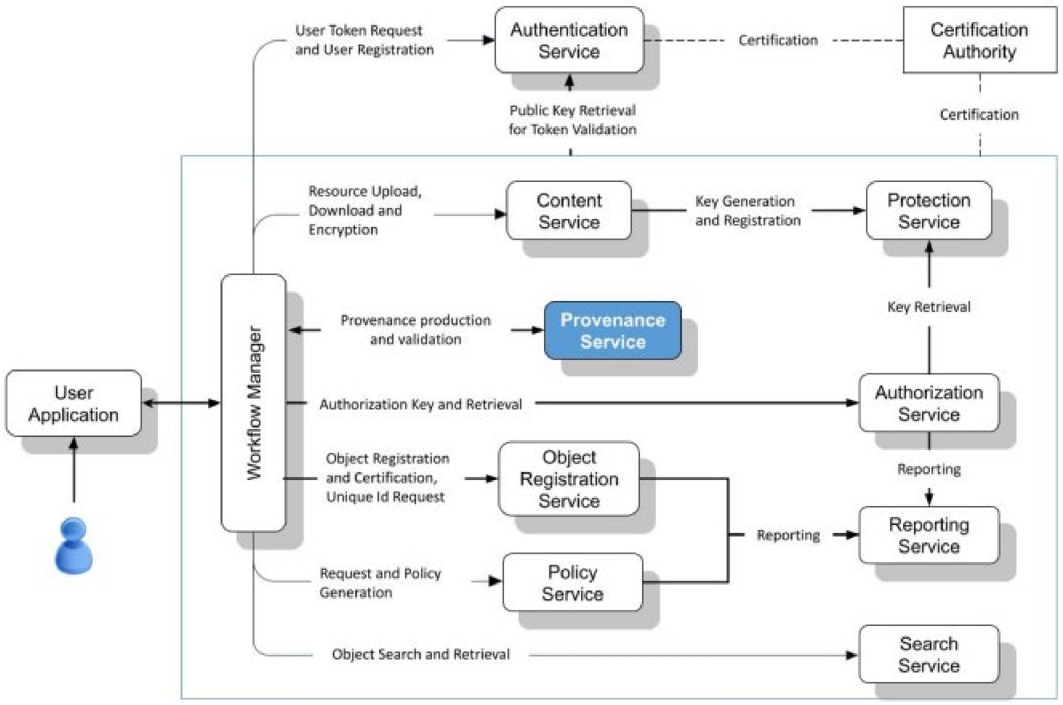

User Token Request and User Registration
Authentication Service
Certification
Certification Authority
Public Key Retrieval for Token Validation
Certification
Resource Upload, Download and Encryption
Content Service
Key Generation and Registration
Protection Service
Key Retrieval
Workflow Manager
Provenance production and validation
Provenance Service
User Application
Authorization Key and Retrieval
Authorization Service
Reporting
Object Registration and Certification, Unique Id Request
Object Registration Service
Reporting
Reporting Service
Request and Policy Generation
Policy Service
Object Search and Retrieval
Search Service

Figure 16: MIPAMS architecture enhanced with Provenance Service

4. Requirements compliance table

The "Requirements compliance table" is attached as an Excel file. Table 4 summarizes the considered requirements. The rest of this section explains how the proposed specification covers the set of JPEG Fake Media requirements listed in the table.

Table 4: Summary of the Requirements compliance table

Req. Nr	Description	Addressed
R1.1	The standard shall provide means to describe how, by whom, where and/or when the media asset was created and/or modified.	yes
R1.2	The standard shall provide means to reference the asset(s) on which the modifications were applied and/or that were used for its creation.	yes
R1.8	The standard shall provide means to keep track of the provenance of media assets and/or of specific modifications.	yes

R2.2	The standard shall comply with the JPEG Systems framework and should retain backwards compatibility.	yes
R2.3	The standard shall consider privacy of individuals and locations.	yes
R2.5	The standard shall provide means to explicitly denote anonymous, obscured, or redacted information. If the information is not provided, then it is considered anonymized.	yes
R2.7	The standard shall be viable as a self-contained structure.	yes
R2.8	The standard shall provide means to verify the integrity of the media asset by:	partially
R.2.8.1	Supporting various hashing methods.	yes
R.2.8.2	Supporting various signing methods.	yes
R2.10	The standard shall be compliant with JPEG Privacy and Security to provide means to secure media asset metadata, including provenance information.	yes
R2.11	The standard shall provide means to provide conditional access to media asset metadata.	yes
R2.13	The standard shall provide means to embed references to externally hosted descriptions, methods and services.	partially
R3.3	The standard shall provide means to verify the authenticity of media assets.	yes
R3.5	The standard shall provide meanest to verify the integrity of media assets.	yes

First and foremost, with the definition of various Assertions and the ability to specify additional metadata around them (e.g., date, metadata, software agent fields), the proposed specification meets requirement R1.1 related to providing description about the modifications that are taking place in a digital asset.

Secondly, the specification provides the Content Binding Assertion as well as the Ingredient Assertion notions in order to reference both the asset that is under modification as well as the ingredient assets that were used to create it (R1.2).

Furthermore, it offers the ability to keep track of the provenance history of a digital asset either by embedding the Manifest Store Content type JUMBF box in the digital asset itself or by storing it in a Manifest Repository on the cloud (R1.8).

The entire data model proposed in this specification is built on top of the ISO/IEC 19566-2022 standard (JUMBF). Thus, by definition, it complies with the JPEG Systems framework (R2.2) and supports the storage of provenance information in a self-contained structure, outside of the referenced digital asset (R2.7). This also partially addresses requirement R2.13, since the Hashed URI references described in subsection 2.1.3.1 supports the reference of JUMBF Boxes that is hosted externally to the JUMBF box specification could reference externally hosted

JUMBF structures. For instance, the Assertion Store Content type JUMBF box could be hosted externally to the Manifest JUMBF box structure.

By allowing Protection Content Type JUMBF Boxes to describe a protected (e.g., encrypted) Assertion, the proposed specification is compliant with ISO/IEC 19566-4: Privacy & Security standard (R2.10). Moreover, it takes into consideration the privacy of both individuals and locations by allowing Assertions related to Exif metadata to be protected (R2.3). Subsequently, the proposed specification offers the choice to a provenance producer to redact a subset of Assertions (e.g., Exif metadata) (R2.5).

When the need of a cryptographic tool is required - e.g., computing digital signatures or calculating the hash of a JUMBF box - the algorithm that is used is also recorded. This is done in order to allow for the adoption of more than one hashing and signing methods (R.2.8.1 and R.2.8.2).

Regarding the protection of privacy related data inside the provenance information, the specification offers the possibility to protect specific Assertions (or part of them) by including them in a Protection Content type JUMBF box. As demonstrated in the use case described in section 3, it is possible for a Provenance Producer to include access rules along with the protected Assertion in order to ensure its conditional access (R2.11).

Finally, related to the proposed provenance operations for producing and consuming provenance information, it is part of the specification to verify the integrity and authenticity of the referenced digital asset through the content binding Assertion which is required in every standard manifest (R3.3 and R3.5).

References

[1] C2PA. [Online]. Available: https://c2pa.org/.

[2] ISO/IEC, "ISO/IEC JTC1SC29/WG1 N100157, Call for Proposals for JPEG Fake Media," 2022.

[3] C2PA, "C2PA Technical Specification v1.0," 2022.

[4] ISO/IEC, " ISO/IEC IS 19566-4:2020 Information Technologies - JPEG Systems Part 4: Privacy and Security," 2020.

[5] ISO/IEC, " ISO/IEC DIS 19566-5:2022, Information Technologies - JPEG Systems - Part 5: JPEG Universal Multimedia Box Format (JUMBF)," 2022.

[6] IETF, "RFC 8949: Concise Binary Object Representation (CBOR)," 2020.

[7] IETF, "RFC 8259: The JavaScript Object Notation (JSON) Data Interchange Format," 2017.

[8] ISO/IEC, "ISO 16684-1:2019, Graphic technology — Extensible metadata platform (XMP) — Part 1: Data model, serialization and core properties," 2019.

[9] JEITA, "JEITA CP-3451: Exchangeable image file format," 2002.

[10] OASIS, "eXtensible Access Control Markup Language (XACML) Version 3.0," 2017.

[11] IETF, "RFC 8152: CBOR Object Signing and Encryption (COSE)," 2017.

[12] W3C, "Verifiable Credentials Data Model v1.1," 2022.

[13] N. Fotos and J. Delgado, "ISO/IEC JTC1SC29/WG1 M96014, UPC Proposal for JUMBF (ISO/IEC 19566-5) Ed2 Reference Software.," 2022.

[14] N. Fotos and J. Delgado, "ISO/IEC JTC1SC29/WG1 M96015, UPC Proposal for JPEG Systems Privacy and Security (ISO/IEC 19566-4) Reference Software v2".

[15] Oracle, "Java Cryptography Architecture (JCA) Reference Guide".

[16] J. Delgado and S. Llorente, "Improving Privacy in JPEG images," in *Proceedings of the IEEE International Conference on Multimedia & Expo Workshops*, Seattle, CA, 2016.