Coalition for Content Provenance and Authenticity

Attestations in the C2PA Framework

0.1, 2023-06-09: Release

Table of Contents

Chapter 1. Introduction

Attestation is a key security feature that enables relying parties, to deduce whether a given entity is trustworthy (example in a correct state and is operating as intended). The given entity can be an application, a platform or a composite device that is been attested.

This specification is a proposal to introduce attestation to C2PA Eco System, i.e. C2PA trust model and its data structures.

C2PA seeks feedback on this proposal, especially from implementers. The underlying draft will be refined based on inputs. The goal is to either refine a non-draft specification or a revision to the main specification.

This draft assumes a working understanding of the C2PA Architecture and Data Structures: in particular, Claims, Assertions, Claim Signatures and how these data objects are packaged into C2PA manifests.

Chapter 2. Relevance to C2PA

There are scenarios where introducing attestation within C2PA Eco-System further enhance the overall security posture of the system. The below examples are few use cases.

2.1. Trusted Camera Application on an Android Phone

Sophisticated and easy to use media editing tools as well as synthetic image generation technologies are widely available. The existence of these tools degrades the probative value of all digital imagery.

A Trusted Camera application can use attestation to

1. indicate the application that captured the image,
2. implicitly prove that an image was obtained from the sensor,
3. indicate that the phone operating system is up-to-date and configured securely.

While no system is foolproof and immune from security threats, for some scenarios an attestation can markedly increase the likelihood that an image is of a real scene.

Android Play Integrity is a Google service for obtaining such attestations. We expect (but do not require) that the key used to create the Claim Signature is associated with the owner/user of the camera phone.

2.2. AI/ML Models running in a container or Isolated Virtual Machine

Machine learning models are increasingly responsible for security-critical tasks, and training sets, models, and inferences will all benefit from C2PA-style provenance signals and protection. In some cases, ML models execute in environments with poor or unknown security characteristics. In these cases, attestations can be added to demonstrate that a model or inference was trained/run in a secure environment.

For concreteness, we assume an ML image recognition model with a valid C2PA manifest running in an SGX enclave, which then outputs metadata - information about any recognized objects - with an asset manifest that references the input image and the model that produced the information about the objects. The resulting manifest is signed by the claim creator (as normal), but also includes an attestation that indicates the code running in the enclave that performed the image recognition steps. Relying parties can use the manifest and attestation assertion to ensure that the image was processed by an authorized tool or model.

Chapter 3. Specifics of Attestation

Attestation is (typically) a specialized type of signing operation where the set of claims about the platform and application (that requested attestation), is signed by the trusted authority to generate attestation token, also known as Evidence in attestation eco-system.

For example:

- TPM quote-operations are a signature over some application-provided data together with an encoding of current Platform Configuration Registers (PCR) values. Compliant platforms record the booting software into the PCRs, so the TPM quote reports reflects the software running on the platform.
- Android Play Integrity is a system/cloud-service that allows app-developers to generate signed statements indicating the package that requested the integrity verdict, as well information about the security posture of the requesting device.
- Intel's Software Guard Extensions (SGX) provides a reporting/quoting feature that is a signature over the measurement of the code that was loaded into the enclave, as well as other critical security settings.
- ARM's Platform Security Architecture (PSA) provides the calling application to collect an attestation token (which is a signature over a set of claims about underlying platform with a supplied freshness parameter from application).

Attestations can be directly consumed by relying parties or can be forwarded to a Trust Broker (independent Verification Service). For example, Android Play Integrity verdicts use a Google-provided Trust Broker service.

Chapter 4. Scenarios

This document is motivated by two specific scenarios. We hope and expect that the architecture will support additional scenarios and attestation technologies. C2PA welcomes feedback if this is not the case.

4.1. Trusted Camera Application on Android Phone

Sophisticated and easy to use media editing tools as well as synthetic image generation technologies are widely available. The existence of these tools degrades the probative value of all digital imagery.

4.2. More Text Here

Chapter 5. Implicit Attestation

This section describes the design considerations and normative requirements for incorporating implicit attestations into the C2PA trust model.

Implicit Attestation (sometimes called Key Attestation), in the context of this specification, involves the use of a signing key that can only be used by an authorized application running on an authorized device. If the signing key can only be used by the authorized application on the authorized devices, then the presence of a well-formed signature in a manifest *implies* that the authorized application signed the Claim. I.e., the Claim-Signing application has been *implicitly attested*. Keys with these security properties are called Implicit Attestation keys, or *IA-keys*.

The IA key is used to sign a Claim in exactly the same way as Claims are normally signed. I.e., the signature is calculated over the serialized Claim, and the resulting Claim Signature is packaged in the manifest identically to other Claim Signatures.

This specification also allows an IA-key signature to be embedded in an attestation assertion as an alternative to incorporating it as a Claim Signature. This is described in Section 6.5.

5.1. Implicit Attestation Requirements

The current (v1.3) version of the C2PA supports the creation and validation of Implicit Attestation Claim Signatures without changes:

> Identity of Signers
>
> The identity of a signatory is not necessarily a human actor, and the identity presented may be a pseudonym, completely anonymous, or pertain to a service or trusted hardware device with its own identity, including an application running inside such a service or trusted hardware.

IA-keys and certificates are generally associated with devices, as opposed to people or organizations. Claim Creators may use the `CreativeWork` assertion to encode a person or organization or can embed the Implicit Attestation in an assertion, as described in Section 6.5.

5.2. Issuing / Certifying Keys for Implicit Attestation

This specification provides guidance but no normative requirements for the issuance and lifetime management of implicit attestation keys. Implementers should consult platform-specific documentation for detailed protocol descriptions.

Devices that support attestation provide cryptographic building blocks that can be used in protocols to prove the identity of the device and running software to relying parties. One common cryptographic primitive is usually called *Quote*. Quotes are digital signatures over user-supplied data and platform/quote-engine-supplied data that describes

the running program and the security properties of the environment in which it is executing.

Quotes (and related primitives) can be used to create Explicit Attestations (Chapter 6) but can also be used in a protocol to provision or certify an Implicit Attestation key. A commonly used protocol fragment is that an application creates a key-pair and "Quotes" the resulting public key. The public key and quote is sent to a trusted service that checks that the device is known or in good-standing, and that the program measurements conveyed in the quote are in policy. If the checks succeed, the service creates a certificate for the newly created Implicit Attestation public key.

(Note that this description is simplified: implementers should consult platform documentation for recommended provisioning protocols for Implicit Attestation keys.)

Once provisioned, the Claim Creator uses the corresponding private key to sign the serialized Claim (Claim-Signer Implicit Attestation) or sign a serialize Partial Claim (Embedded Implicit Attestation.)

5.3. IA-Key Certificates

The certificate for the implicit attestation claim signing key should comply with the requirements in the main C2PA specification. Optionally, the certificate may contain additional fields that convey additional information about the application or security posture of the device to which it was provisioned.

5.4. Protecting Implicit Attestation Signing Keys

Most platforms that provide attestation capabilities also provide security primitives to allow a system to protect stored keys and other data so that the data is only accessible to the attesting application, or other applications explicitly authorized by the attesting application. This operation is commonly called *sealing*.

Implementers should consult platform documentation for usage and the expected strength of protection. If suitable sealing primitives are not available, applications should not persist IA-keys: instead they should re-provision on every app-startup (or use Explicit Attestation).

5.5. Timeline for Provisioning and Use of Implicit Attestation Signing Keys

Summarizing the previous sections in the form of a timeline, the following steps are required when provisioning and using a key for implicit attestation.

1. It is assumed that devices are provisioned with a platform key and certificate (or a key and a pre-established trust relationship with a server.) This is usually performed during manufacture.
2. At some point in the life of the device, the C2PA-compliant Claim Creator application is installed.
3. During installation (or later), the Claim Creator will interact with a trusted service to create a certified key for signing Claims. The Claim Creator uses Quotes (or similar) to prove that it is an authorized application running on a trusted device. The Quote directly or indirectly uses the platform key in (1).

4. The certified IA-key and certificate can be used immediately to sign Claims or Partial Claims.
5. If the platform provides suitable *sealing* facilities, the IA-private key can be persisted in such a way that only the authorized application can retrieve it.
    a. If this is the case, then the IA-key can be stored and protected so that the application be closed and restarted and use the same key
    b. If this is not the case, then the IA-key should not be persisted: instead, the application should follow step (3) to obtain a new IA-key on each startup.

5.6. Claim Validation of Manifests Using Implicit Attestation

Claim-Signer Implicit Attestation: Claims signed with IA-keys are validated identically to validation of manifests signed with keys associated with organizations or people.

We expect that IA-keys will employ specialized CAs and Trust Roots, although this is not required.

:sectlinks :xrefstyle: short :sectnumlevels: 5

Chapter 6. Explicit Attestation

In this section we describe how an attestation can created and embedded as an assertion. This contrast with the architecture described in the Implicit Attestation section where the (implicit) attestation is conveyed as a Claim Signature. Implementation considerations for choosing Implicit vs. Explicit Attestation is discussed in section Section 6.1.

The current C2PA trust model is built using two signatures:

1. A signature over a CBOR-serialized Claim. The signature is encoded in the manifest, and the certificate chain for the signing key is usually also included.
2. Optionally, a countersignature from an RFC 3161-compliant time stamping service.

Attestations, in the context of this document, are also digital signatures. This section describes options for how the additional attestation signature(s) can be added to the two that are already defined. Note that the discussion in this section is simplified: later sections present the complete design and requirements.

In the following, when we say *"attestation over X"* we mean that the attestation is a signature over the hash of the serialization of X.

The simplest option for adding an attestation would be to include it as a second countersignature over the Claim. However, with this design, the Claim Signature or the attestation can be independently removed and replaced. Whether these attacks are troublesome will depend on scenario (and the sophistication of the relying party), which makes a thorough security analysis difficult. However, the following proposal prevents the attestation and claim signatures being independently replaced with only modest complexity.

To achieve the Claim-Signature/attestation binding, an attestation over the serialized claim is performed first and then the claim signature is performed over the attestation. This prevents the attestation being removed or replaced independently of the Claim Signature, because any modification of the attestation will invalidate the Claim Signature

To prevent the Claim Signature being replaced, the attestation can optionally contain the public key of the subsequent Claim Signer. This means that validators can detect and reject a Claim Signature if the Claim Signature key specified in the attestation does not match the keys used for the actual Claim Signature.

To summarize: The Claim Signer signs the attestation, which prevents tampering of the attestation. The attestation contains the public key of the Claim Signer, so the Claim Signature cannot be replaced with a signature using a different key. Note that there are no cryptographic protections against both the attestation and Claim Signature being stripped or replaced together.

The binding is illustrated in Figure 1.

Figure 1. Cryptographic dependencies for an Asset, Claim, Attestation and Claim, Signature. An arrow pointing from A to B means that A cryptographically depends on B, so B is integrity protected. (In subsequent sections we will introduce Partial Claims, which changes the diagram but not the dependencies.

6.1. What is Attested?

Claim Signatures are calculated over the serialization of a Claim, and the Claim itself is cryptographically linked to its embedded Assertions. Two types of Assertion are important here: Assertions that contain metadata, and Assertions that bind a Claim to an Asset with a hash or hashes of an Asset or Asset fragment. By signing the Claim, the Claim Creator vouches for both the asset (via the binding Assertion) and the metadata.

We desire similar bindings for the explicit attestation signature: i.e., the attestation vouches for the asset and the metadata. Note that most or all scenarios benefit from binding to the Asset (e.g., for the Trusted Camera Application, *"this image was captured by this program running on this device."*) Some scenarios will benefit from binding to metadata assertions (e.g., for a Trusted Camera Application *"the GPS coordinates obtained from the radio when this image was captured"* or for the ML scenario, *"the following objects were recognized in the image."*) However, note that attestation will *not* improve trust for all types of assertion data: for example, user-input is hearsay as far as the attesting application is concerned.

The approach we have taken is that the attestation is over the serialization of the entire Claim – i.e., the same data structure that the Claim Generator signs (with one exception, which we describe below.) As noted above, not all assertions/metadata gain security benefit from the attestation, but signing all assertions provides the most flexibility for future scenarios. For future advanced use cases, we define a field that an attestation generator can use to indicate which assertions are meaningfully "attested." However, the use and interpretation of this field is beyond the current scope of C2PA.

6.2. Embedding an Explicit Attestation in a Manifest

Explicit attestations are similar to Claim Signatures and could be encoded in a manifest similarly. However, we also have the additional security requirement noted earlier: i.e., that the Claim Signer should also sign the attestation result.

A natural way of encoding the attestation so that it is signed by the Claim Signer is to embed it as an assertion in the Claim. If we do this, then the attestation will naturally be signed in the same way as any other sort of Assertion. However, naïve attempts at implementing this will fail because we are introducing a circular dependency: the Claim cannot be finalized before the attestation is created, but the attestation requires a finalized Claim in order to calculate the Claim hash.

We choose to break this dependency by specifying that the attestation is created over the hash of the serialized claim

but **omitting the attestation assertion** (which has not been created yet.) Once the attestation has been created, it is embedded in the Manifest and Claim identically to any other assertion and will subsequently be signed by the Claim Creator. The Claim with one or more attestation assertions elided is called a Partial Claim.

This architecture is attractive because normal Claim creation and Claim Validation are unaffected. It is also attractive because an attestation-enhanced claim can be processed by an attestation-unaware validator without changes (the attestation assertion can be treated like any other third-party or unrecognized assertion.)

The (simplified) logical flow for creating a manifest with and without an attestation is illustrated in Figure 2.

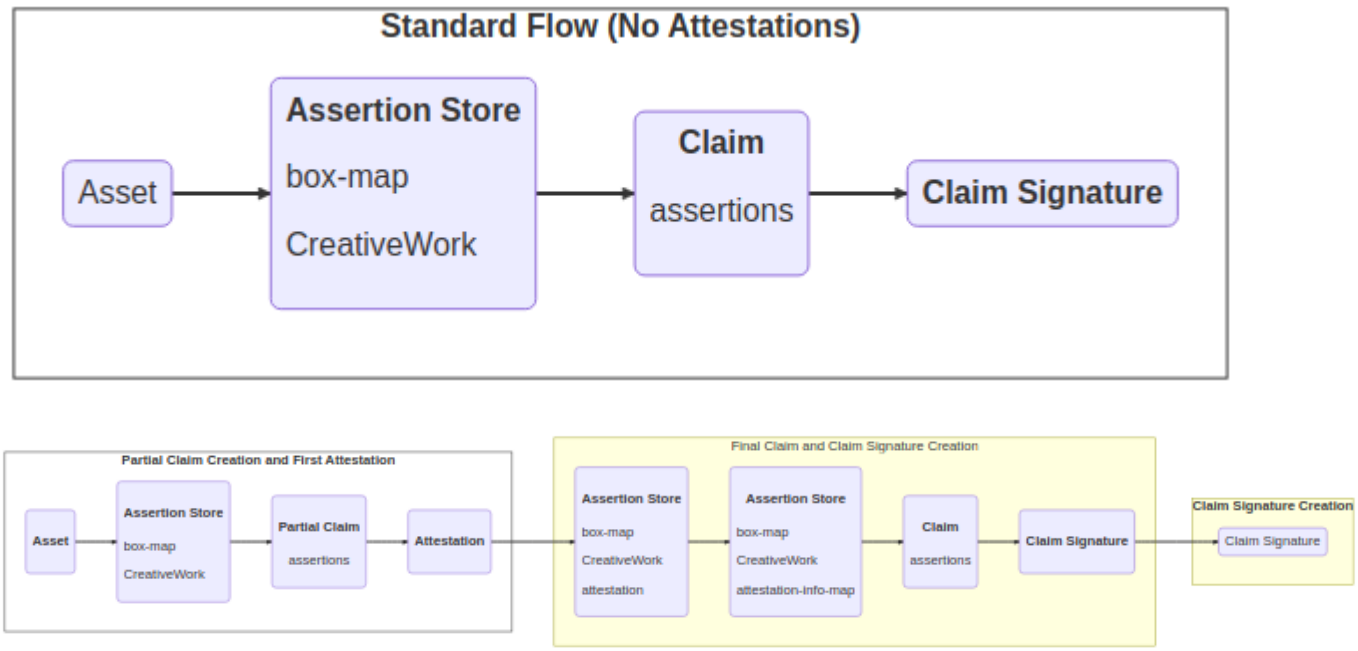

Figure 2. Simplified steps for creating a C2PA Manifest and a Manifest containing an Attestation Assertion. In the no-attestation case, Assertions are created and boxed in the Assertion Store. A Claim is then prepared, with the `assertions` *array set to the location and hash of the referenced assertions. The Claim is then serialized, hashed, and signed by the Claim Creator. The Assertion Store, Claim, and Claim Signature are then packaged as a manifest (not shown.) To support attestations, the additional steps in the shaded box are required. As before, an Assertion Store is populated with the desired assertions, but we call it a Partial Assertion Store because one additional Assertion will be added before the Claim is finalized. The Partial Claim is then serialized and Claim hash is then attested using the appropriate platform attestation service. Next, the Attestation is packaged as an Assertion and added to the Partial Assertion Store to create the Finalized Assertion Store. Finally, the serialized-Claim (with the embedded Attestation Assertion) is signed by the Claim Creator.*

Note that this architecture *does* demand that attestation-aware Validators perform additional steps: Specifically, Validators must edit the Claim to remove any attestation assertions, and then re-serialize the resulting Partial Claim to validate the binding in the attestation assertion. We consider this tradeoff to be acceptable: the rewriting cost is borne by the creators and consumers of attestations, but the creation and validation workflow for non-attestation-aware entities is unchanged.

Additional details are included in the normative parts of the specification below.

6.3. Cryptographic Dependencies for the Asset, Partial Claim, and Claim Signature

The Figure 3 is a more detailed version of Figure 1 using Partial Claims

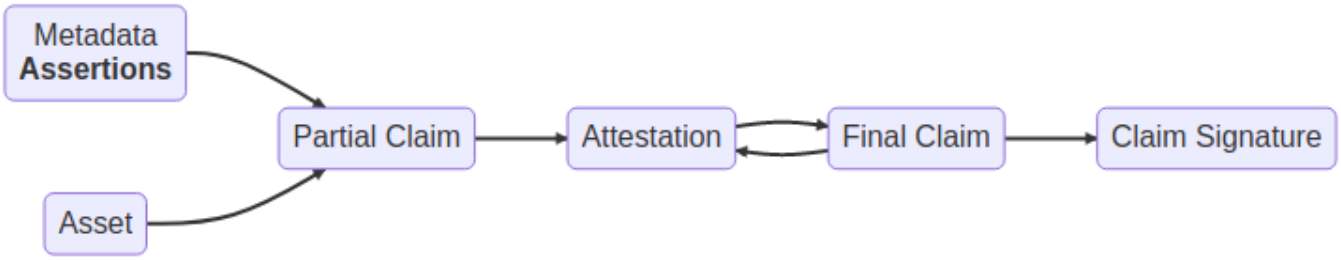

Figure 3. Cryptographic dependencies for the asset, Partial Claim, and Claim.

As before, an arrow from A to B indicates that B cannot be changed without invalidating the manifest. Practically, this means that the attestation and Claim Signature can both be stripped and replaced, but neither can be changed independently.

6.4. Multiple Explicit Attestations

This specification supports more than one attestation for a Claim – for example, there may be one attestation for code running in an enclave, and a second for the overall platform (the “rich OS.”)

If more than one attestation is required then they are ordered and each subsequent attestation is over the Partial Claim containing the prior attestations. For example, if two attestations are required, the first attestation to be added will be over the Partial Claim with no attestation assertions. The first attestation assertion is then added to the Claim, and the second attestation will be over the claim with just the first attestation assertion included. Of course, the second attestation assertion is then added, and the resulting finalized Claim is signed by the Claim Creator.

The cryptographic dependencies are illustrated in Figure 4. The manifest is built from the top to the bottom. An arrow from A to B indicates that B has a cryptographic dependence on A.

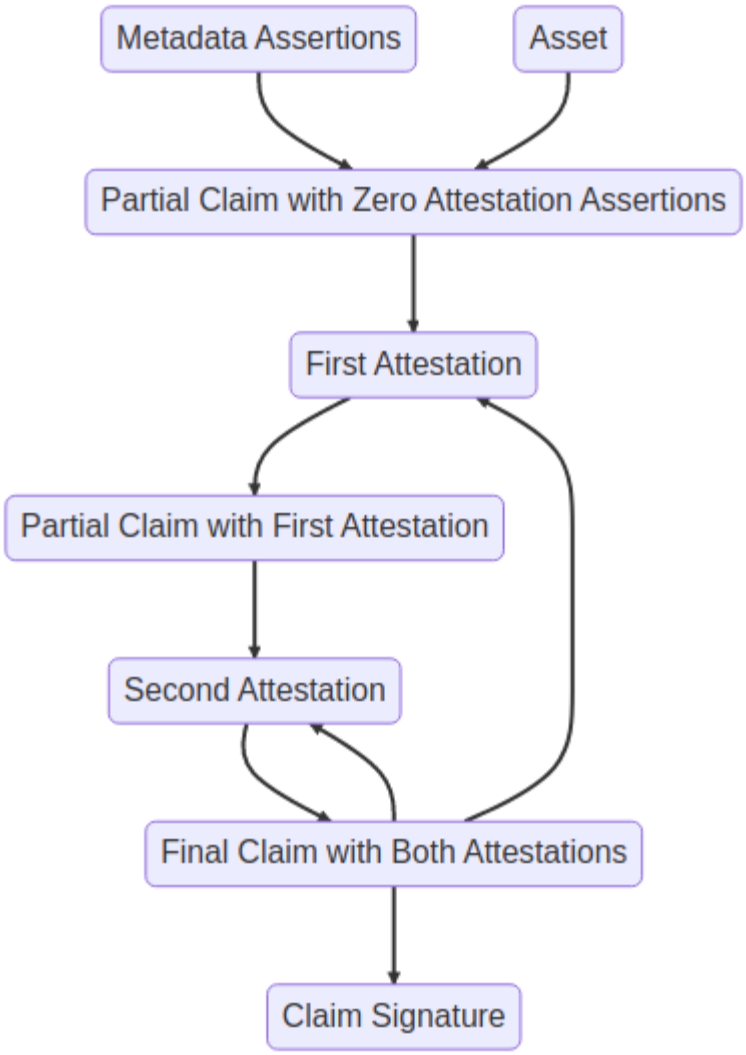

Figure 4. Cryptographic dependencies for a manifest with two explicit attestations.

Attestors that wish to include multiple attestations are free to decide the order that they are created and embedded, but since later attestations include earlier attestations, the validation order must be the same as the creation order. To ensure that that this occurs, this specification requires that attestations are embedded in the assertions array in the order that they are created.

6.5. Implicit Attestation Assertions

Chapter 5 describes Implicit Attestation in detail, but briefly, an Implicit Attestation is a simple signature (i.e., not a *"quote"*) using a key that is only accessible to authorized applications running on authorized devices. If the key can only be used by trusted applications, then the presence of a well-formed Claim Signature *implies* that the trusted app signed the Claim.

Section 5.1 describes how Implicit Attestations can be encoded as Claim Signatures. This is simple and straightforward, but C2PA does not support multiple Claim Signatures in a Manifest so may not be suitable when the Claim Creator needs to convey both the (organization- or human-) Claim Signer *and* the application+device on which the Manifest was created.

To remedy this, the specification also allows Implicit Attestations to be encoded in Attestation Assertions, in exactly the same way that Explicit Attestations are supported, with the only difference being that an Implicit Attestation is a "simple" signature, whereas an Explicit Attestation is typically a quote-style signature.

Attestation Assertions for Implicit Attestations are distinguished using the type field. Implicit Attestations use the attestation type `c2pa.embedded-implicit` and the actual signature is embedded in the `attestation-info-map.att-result` field. Details are provided in Appendix A.

6.6. Normative Requirements for Explicit Attestations

6.6.1. Data Structures

6.6.1.1. Partial Claim Definition

A Partial Claim is a C2PA `claim-map` but omitting any Attestation Assertions. (Note: for simplicity we say that an assertion is “included” or “omitted”, but what is actually added or omitted from the Claim is actually a `hashed-uri-map` link in the `assertions` array.) Partial Claims use the same `claim-map` data structure as a standard Claim: a Partial Claim only differs in that one or more Attestation Assertions are removed from the `assertions` array.

Attestation Assertions can be included at any location in the assertion array, although placing them last in the assertions array simplifies processing and is therefore preferred.

During Claim creation, the attestations are calculated then embedded one at a time. For example, the Partial Claim without attestation assertions is created, serialized, and then the first attestation is gathered. The resulting attestation result is then encoded as an `attestation-info-map` assertion and then added to the end of the `assertions` array in the Partial Claim form the finalized Claim.

If a second attestation is required, the Partial Claim with the first Attestation Assertion included is serialized and the second attestation is performed. The second Attestation Assertion is encoded in a `attestation-info-map` then added to the end of the `assertions` array to form the final Claim, which is then signed by the Claim Creator.

This specification does not restrict the location in the `assertions` array where the Attestation Assertions are added,[1] although, as previously noted, incorporating them last in the array is preferred. However, the order in the `assertions` array is important when more than one Attestation Assertion is incorporated. In this case, higher-index Attestation Assertions are performed after lower-index entries.

6.6.1.2. Attestation "To-Be-Signed" Definition

The `attestation-tbs-map` (attestation to-be-signed map) is an envelope data structure for the information that is to be serialized, hashed, and signed by the attestation machinery. The most important field in `attestation-tbs-map` is the `partial-claim-hash`: essentially the hash of the Asset and its referenced Assertions.

```
; Container data structure for the information "attested"
```

```
attestation-tbs-map =
{
    “partial-claim-hash”: bstr,     ;hash of partial claim
    “alg”: tstr                     ;hash algorithm used in part-claim-hash
    ? “pub-key”: bstr,              ;claim signer public key
    ? “created”: time,              ;UTC when this map was created
    ? “other-tbs-info”: tstr        ;Optional parameters
    ? “other-tbs-info-2”: tstr      ;Optional parameters
}
```

Field	Description
`partial-claim-hash`	The hash of a Partial Claim – i.e., a Claim that omits all or some of the Attestation Assertions. If the Claim contains a single Attestation Assertion, then this will be the hash of the CBOR-serialized Claim omitting the Attestation Assertion – i.e., usually the final entry in the `assertions` array. If the claim contains *n* Attestation Assertions, then *n* Partial Claim hashes are defined, with zero to *n-1* embedded Attestation Assertions.
`alg`	The hash algorithm used to compute `partial-claim-hash`.
`pub-key`	(Optional) The DER-encoded public key and algorithm of the Claim Signer (the Subject Public Key as used in the X.509 certificate) as specified in RFC-5480 and RFC-8017.
`created`	(Optional) The UTC time that the attestation was created. Can be set to reduce the risk that the same claim signer can later add a different claim signature.
`other-tbs-info`	(Optional) Additional information that the Claim Creator wishes to associate with the attestation.
`other-tbs-info-2`	(Optional) Additional information that the Claim Creator wishes to associate with the attestation.

6.6.1.3. Attestation Assertion Definition

`attestation-info-map` is the envelope data structure that contains the attestation data/signature and related information. It is used to embed an Attestation Assertion in the Assertion Store.

`attestation-info-maps` are boxed with label `c2pa.attestation` (or `c2pa.attestation`, `c2pa.attestation_001`, `c2pa.attestation_002`, if more than one attestation is required.)

Attestation Assertions can be placed at any index in the `assertions` array but adding them as the final entries in the order that they are created is preferred.

The `attestation-info-map` definition presented here can encode many types of attestation. See Appendix A for currently defined attestation encodings.

```
; Encoding of an attestation into an assertion

attestation-info-map = {
```

```
  "att-type":tstr,                          ; type of attestation
  “attestation-tbs”: attestation-tbs-map,   ; The attestation-tbs-map
  "attestation-results": bstr,              ; attestation result/signature
  "certificates" : tstr,                  ; PEM-encoded certificate or certificate chain
  ?“other-info”: bstr                       ; other info
  ?“other-info-2”: bstr                     ; other info
  ?“created”: time                          ; attestation creation time
}
```

Field	Description
`att-type`	The attesation type. E.g. `c2pa.SGX`, See Appendix A for currently defined types. If non-standard attestations are encoded, the same convention for non-standard Assertion labels should be used. E.g., `com.litware.custom-assertion`
`attestation-tbs`	The `attestation-tbs-map` used when the assertion was obtained.
`attestation-results`	Binary encoding of the attestation measurement.
`certificates`	(Optional) string version of the PEM encoded certificate of certificate chain for the attesting key.
`creation-time`	(Optional) UTC time when the attestation was created.
`other-info`	(Optional) Any additional information that the Claim Creator wishes to associate with the attestation.
`other-info-2`	(Optional) Any additional information that the Claim Creator wishes to associate with the attestation.

6.7. Creating a Claim Containing One or More Explicit Attestations

These are the detailed steps for creating an embedding an Attestation into a Claim. In the 1.3 version of the C2PA specification, these steps should be performed immediately prior to *“11.3.2.4. Signing a Claim”*. Following the steps listed here, the Claim should be signed as described in the main specification.

1. Create a copy of the `claim-map` containing all non-attestation assertions.
2. Calculate the hash of the serialized (Partial) `claim-map` to form *hash(claim-map)*.
3. Create an `attestation-tbs-map` data structure, including *hash(claim-map)*, the time, and (optionally) the public key of the Claim Signer. Add any desired optional data to `attestation-tbs-map`. CBOR-serialize and hash to form *hash(attestation-tbs-map)*.
4. Use the appropriate platform service to obtain an attestation over *hash(attestation-tbs-map)* to form attestation *attest(hash(`attestation-tbs-map`))*
5. Create a new `attestation-info-map`, then:
    a. Set `att-type` to the type-selector for the attestation performed (Appendix A)

b. Set `attestation-tbs` to `attestation-tbs-map`

c. Set `attestation-results` to *attest(hash(attestation-tbs-map))* obtained in the previous step. Note that defined types and encodings are specified in Appendix A.

6. Create a JUMBF attestation container with label `c2pa.attestation` (or `c2pa.attestation_00n`, if more than one attestation must be included) containing the `attestation-info-map` being prepared.
7. Add the `attestation-info-map` to the Assertion Store.
8. Add a new `hashed-uri-map` to the `assertions` array in `claim-map`, referencing the newly added Attestation Assertion.
9. If more than one attestation is required, go back to (2), but start with the current Partial `claim-map` containing the attestations calculated thus far.

At this point, the final claim has been created, and processing can continue according to *"11.3.2.4. Signing a Claim"* using the `claim-map` with all embedded Attestation Assertions.

Note that this generic flow does not specify how an attestation is encoded into `attestation-results`. This is scheme-specific and is described in Appendix A.

6.8. Validating a Claim Containing Explicit Attestations

This section describes how Claims containing explicit attestations should be validated. This section also describes how non-attestation aware validators should behave: this text will be replicated in the main C2PA technical specification.

6.8.1. Attestation Aware Validator

PRELIMINARY: MORE DETAIL NEEDED.

To perform attestation validation, first all Attestation Assertions are removed from the `assertions` array in the Claim. The first Attestation Assertion is then checked against the hash of the CBOR-serialized Partial Claim with all Attestation Assertions removed. If this succeeds, then the first Attestation Assertion is added back into the `assertions` array, the resulting Partial Claim is re-serialized, and the second Attestation Assertion is validated, and so on. Note that for validation to succeed, the Attestation Assertions must be re-added in the same location in the assertions array. This is simpler if they are last in the `assertions` array, which is the reason for this recommendation. However, attestation validators should be able to process Attestation Assertions in any location in the array.

Note that when an Attestation Assertion is removed, the Claim is serialized using normal CBOR rules. For example, if a claim included two standard assertions and one Attestation Assertion, then the Partial Claim will contain an `assertions` array containing two elements. If a Partial Claim contains two standard assertions and two Attestation Assertions, then two Partial Claims are defined: one omitting just the last Attestation Assertion in the `attestations` array, and one omitting both Attestation Assertions. Of course, the order of the Attestation Assertions must be preserved as the Partial Claim is created and validated.

NOTE do pub-key checks, etc.

6.8.2. Non-Attestation Aware Validator

(This text belongs in the main specification.)

This text will be added to the current main-specification validation section. It ensures that assets including attestation assertions compliant with this specification are not rejected.

"For validators that do not consume attestations, any assertion with label starting with `c2pa.attestation` should be ignored. Third party unrecognized attestations, including third-party attestation assertions, are ignored as specified elsewhere."

[1] We do not demand that attestation assertions appear last in the attestation array, because doing so would limit the future evolution of the C2PA standard.

Chapter 7. Implementation Considerations

In this section we describe the tradeoffs when choosing an attestation technology and embedding scheme (explicit or implicit attestations.)

7.1. Attestation Technology

A variety of hardware-based attestation technologies are available. Most platforms offer just one choice, although some platforms have two (e.g., a TPM that can attest the running OS, and SGX enclaves that can attest an app running in the enclave.)

System architects may consider the following factors when choosing an attestation technology:

1. What is available? The platform may only offer one attestation technology. For example, a TPM may be the only attestation technology available on a platform. In this case, the choice is made for the system architect.
2. Which System Services need to be "trusted?" Some aspects of Claim Creation can be performed without use of external services. For example, creating asset-bindings (hashes) can be implemented entirely by the Claim Creator without use of OS services. However, some scenarios benefit from securely attesting data obtained from IO devices such as a camera sensor or GPS receiver. If attestation is to increase trust in data "passed in" from outside the Claim Creator application, then the device or OS that provides this data must be measured, be immutable, or be authenticated.
3. Online or Offline? Some attestation technologies (PlayIntegrity, RATS) require an online connection to a server create an attestation. Others can create attestations offline.

7.2. Implicit or Explicit Attestation

7.3. RATS or Native

7.4. Space and Time Considerations for Attestation Creation

7.5. Space and Time Considerations for Attestation Verification

Chapter 8. Appendix A - Defined Attestation Schemes and Encodings

This appendix describes the currently defined attestation schemes, and how the attestation information is incorporated into a C2PA manifest.

In all cases, attestations should be calculated 'over' the hash of the serialization of the Partial Claim.

This section is preliminary and just contains the gist. We need to refine and validate with implementations.

8.1. Android Play Integrity

Android Play Integrity is a platform service for determining the identity of an Android application, and whether that application is running on an in-policy Android device.

8.1.1. Obtaining an Attestation

Attestations are typically obtained using:

```
integrityManager.requestIntegrityToken(IntegrityTokenRequest.builder().setNonce(nonce).build
());

IntegrityTokenRequest.Builder.setNonce(NonceString)
```

Where the `NonceString` is the base64-encoded hash of the `attestation-tbs-map` for the Claim.

8.1.2. Definition of Fields in `attestation-info-map` for Android Play Integrity

Field	Value
`att-type`	`c2pa.AndroidPlayIntegrity`
`att-result`	UTF-8-encoding of the null-terminated attestation result
`other-info`	(Optional)

8.2. Intel SGX

8.2.1. Obtaining an Attestation

TBD.

8.2.1.1. Definition of Fields in `attestation-info-map` for Intel SGX.

Field	Value
`att-type`	`c2pa.SGX`
`att-result`	UTF-8-encoding of the null-terminated attestation result
`certificate`	
`other-info`	(Optional)

8.3. TPM 2.0

8.3.1. Obtaining an Attestation

Applications should create C2PA attestations using the `TPM2_Quote` operation with the `qualifyingData` set to the digest of the Partial Claim using the same hash algorithm as Platform Configuration Registers (PCRs) that are used.

The key used for signing should be a **restricted signing key**, and applications should specify a set PCRs that adequately represent the security configuration of the host platform.

Depending on scenario, it may be necessary or desireable to also include/embed a certificate chain for the quoting key.

8.3.2. Definition of Fields in `attestation-info-map` for TPM 2.0

Field	Value
`att-type`	`c2pa.TPM2.0`
`att-result`	Raw binary `TPMT_SIGNATURE` obtained from the TPM without padding.
`certificate`	base-64 encoded PEM-encoded certificate chain for the quoting key.
`other-info`	Raw binary `TPM2B_ATTEST` data obtained from the TPM without padding.

8.4. IETF RATS

8.4.1. Obtaining an Attestation

TBD.

8.4.2. Definition of Fields in `attestation-info-map` for IETF RATS

Field	Value
`att-type`	`c2pa.RATS`

`att-result`	TBD
`certificate`	base-64 encoded PEM-encoded certificate chain for the quoting key
`other-info`	Raw binary `TPM2B_ATTEST` data obtained from the TPM

8.5. Implicit Attestations Encoded in Attestation Assertions

8.5.1. Obtaining an Attestation

The Implicit Attestation signature is a signature over the hash of the Partial Claim as described in section Section 6.6.1.1.

8.5.2. Definition of Fields in `attestation-info-map` for Embedded-Implicit Attestation

Field	Value
`att-type`	`c2pa.embedded-implicit`
`att-result`	Signature over Partial Claim using one of the signature encoding allowed by RFC 5280 (https://www.rfc-editor.org/rfc/rfc5280).
`certificate`	(Optional) base-64 encoded PEM-encoded certificate chain for the Implicit Attestation key
`other-info`	TBD