Koalition für die Herkunft und Authentizität von Inhalten

Inhaltszertifikate

Technische Spezifikation der C2PA

2.2, 01.05.2025:

Inhaltsverzeichnis

Kapitel 1. Einleitung

1.1. Überblick

Angesichts der zunehmenden Geschwindigkeit digitaler Inhalte und der zunehmenden Verfügbarkeit leistungsstarker Erstellungs- und Bearbeitungstechniken ist die Feststellung der Herkunft von Medien entscheidend, um Transparenz, Verständnis und letztlich Vertrauen zu gewährleisten.

Wir erleben derzeit außergewöhnliche Herausforderungen für das Vertrauen in die Medien. Da soziale Plattformen die Reichweite und den Einfluss bestimmter Inhalte durch immer komplexere und undurchsichtige Algorithmen verstärken, verbreiten sich falsch zugeordnete und falsch kontextualisierte Inhalte schnell. Ob unbeabsichtigte Fehlinformationen oder bewusste Täuschung durch Desinformation – unechte Inhalte nehmen zu.

Derzeit können diejenigen, die Metadaten zu ihren Werken hinzufügen möchten, dies nicht auf sichere, manipulationssichere und plattformübergreifend standardisierte Weise tun. Ohne diese Informationen aus einer anerkannten Quelle fehlt Verlegern und Verbrauchern der entscheidende Kontext, um die Authentizität von Medien zu bestimmen.

Die Herkunftsangabe ermöglicht es Content-Erstellern und Redakteuren, unabhängig von ihrem geografischen Standort oder ihrem Zugang zu Technologie, Informationen darüber offenzulegen, wie ein Asset erstellt wurde, wie es geändert wurde und was geändert wurde. Bei jeder Änderung eines Assets bleibt die bestehende Provenienz des Assets erhalten, wobei jede neue Änderung zur Provenienz hinzugefügt wird. Auf diese Weise liefern Inhalte mit Provenienz Indikatoren für die Authentizität, sodass Verbraucher sich der geänderten Inhalte bewusst werden können. Eine solche Provenienz könnte umfassen, was geändert wurde und woher diese Änderungen stammen. Diese Möglichkeit, Urhebern, Verlegern und Verbrauchern eine Provenienz zu liefern, ist für die Förderung des Vertrauens im Internet von entscheidender Bedeutung.

Um dieses Problem für Verlage, Urheber und Verbraucher in großem Maßstab anzugehen, hat die Coalition for Content Provenance and Authenticity (C2PA) diese technische Spezifikation zur Gewährleistung der Herkunft und Authentizität von Inhalten entwickelt. Sie soll die weltweite, freiwillige Einführung digitaler Herkunftstechniken ermöglichen, indem ein reichhaltiges Ökosystem digitaler Herkunftsanwendungen für eine Vielzahl von Einzelpersonen und Organisationen geschaffen wird, das gleichzeitig den entsprechenden Sicherheitsanforderungen entspricht.

Diese Spezifikation basiert auf Szenarien, Arbeitsabläufen und Anforderungen, die von Branchenexperten und Partnerorganisationen, darunter die Project Origin Alliance und die Content Authenticity Initiative (CAI), zusammengetragen wurden. Es ist auch möglich, dass Aufsichtsbehörden und Regierungsbehörden diese Spezifikation zur Festlegung von Standards für die digitale Herkunft nutzen könnten.

1.2. Geltungsbereich

Diese Spezifikation beschreibt die technischen Aspekte der C2PA-Architektur, einem Modell zum Speichern und Abrufen kryptografisch überprüfbarer Informationen, deren Vertrauenswürdigkeit anhand eines definierten Vertrauensmodells bewertet werden kann. Dieses Dokument enthält Informationen zur Erstellung und Verarbeitung eines C2PA-Manifests und seiner Komponenten, einschließlich der Verwendung digitaler Signaturtechnologie zur Manipulationssicherung und Vertrauensbildung.

Vor der Entwicklung dieser Spezifikation hat die C2PA unsere Leitprinzipien erstellt, die es uns ermöglichten, uns darauf zu konzentrieren, dass die Spezifikation in einer Weise verwendet werden kann, die den Datenschutz und die persönliche Kontrolle über Daten respektiert und potenzielle Missbräuche kritisch hinterfragt. Beispielsweise werden die Implementierer dieser Spezifikation nachdrücklich aufgefordert, den Erstellern und Herausgebern von Medieninhalten die Möglichkeit zu geben, zu kontrollieren, ob bestimmte Herkunftsdaten enthalten sind.

WICHTIG

Aus dem Abschnitt „Übergeordnete Ziele" der Leitprinzipien:

> Die C2PA-Spezifikationen SOLLTEN KEINE Werturteile darüber abgeben, ob eine bestimmte Gruppe von Herkunftsdaten „gut" oder „schlecht" ist, sondern lediglich darüber, ob die darin enthaltenen Aussagen als mit dem zugrunde liegenden Vermögenswert verbunden, korrekt formuliert und frei von Manipulationen validiert werden können.

Es ist wichtig, dass die Spezifikation die Zugänglichkeit von Inhalten für Verbraucher nicht beeinträchtigt.

Andere Dokumente der C2PA befassen sich mit spezifischen Überlegungen zur Umsetzung, wie z. B. den erwarteten Benutzererfahrungen und Details zu unserer Modellierung von Bedrohungen und Schäden.

1.3. Technischer Überblick

Die C2PA-Informationen umfassen eine Reihe von Aussagen, die Bereiche wie die Erstellung von Assets, Bearbeitungsaktionen, Details zu Erfassungsgeräten, Verknüpfungen zu Inhalten und viele andere Themen abdecken. Diese Aussagen, die als Assertions bezeichnet werden, bilden die Herkunft eines bestimmten Assets und stellen eine Reihe von Vertrauenssignalen dar, die von Menschen genutzt werden können, um ihre Einschätzung der Vertrauenswürdigkeit des Assets zu verbessern. Assertions werden zusammen mit zusätzlichen Informationen in einer digital signierten Einheit zusammengefasst, die als Claim bezeichnet wird. Dieser Claim wird vom Claim-Generator im Namen des Unterzeichners unter Verwendung der Signaturberechtigungsnachweise des Unterzeichners digital signiert, wodurch die Claim-Signatur entsteht.

Diese Behauptungen, Ansprüche und die Anspruchsignatur werden alle durch eine Hardware- oder Softwarekomponente, den sogenannten Anspruchgenerator, zu einer überprüfbaren Einheit namens C2PA-Manifest (siehe Abbildung 1, „Ein C2PA-Manifest und seine Bestandteile") zusammengefasst. Die in den Inhaltsnachweisen der Ressource gespeicherten C2PA-Manifeste stellen deren Herkunftsdaten dar.

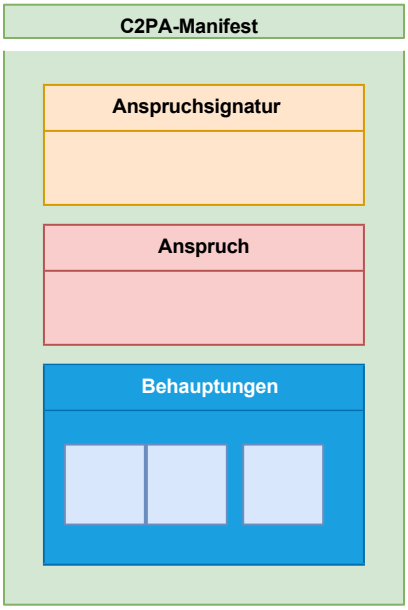

Abbildung 1. Ein C2PA-Manifest und seine Bestandteile

1.3.1. Vertrauen aufbauen

Die Grundlage für Vertrauensentscheidungen in C2PA, unserem Vertrauensmodell, ist die Identität des Unterzeichners, der mit dem kryptografischen Signaturschlüssel verbunden ist, der zum Signieren eines Anspruchs in einem C2PA-Manifest verwendet wird. Die Anspruchssignaturen von C2PA-Manifesten können in Kombination mit vertrauenswürdigen Zeitstempeln unbegrenzt lang einem Validierungsprozess unterzogen werden, um festzustellen, ob die Ansprüche signiert wurden, während die Signaturberechtigungen gültig und nicht widerrufen waren.

1.3.2. Ein Beispiel

Ein sehr häufiges Szenario ist, dass ein Benutzer mit seiner C2PA-fähigen Kamera (oder seinem Smartphone) ein Foto aufnimmt. In diesem Fall erstellt die Kamera ein Manifest, das einige Aussagen enthält, darunter Informationen über die Kamera selbst, eine Miniaturansicht des Bildes und einige kryptografische Hashes, die das Foto mit dem Manifest verknüpfen. Diese Aussagen würden dann in der Forderung aufgeführt, die digital signiert würde, und anschließend würde das gesamte C2PA-Manifest (siehe Abbildung 2, „Beispiel für ein C2PA-Manifest eines Fotos“) in die JPEG-Ausgabe eingebettet werden. Dieses C2PA-Manifest würde auf unbestimmte Zeit gültig bleiben.

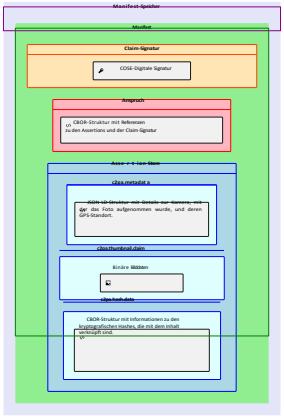

Abbildung 2. Beispiel für ein C2PA-Manifest einer Fotografie

Ein Manifest-Konsument, wie beispielsweise ein C2PA-Validator, hilft Benutzern dabei, die Vertrauenswürdigkeit der Ressource festzustellen, indem er zunächst die digitale Signatur und die zugehörigen Anmeldedaten validiert. Außerdem überprüft er jede der Aussagen auf ihre Gültigkeit und präsentiert dem Benutzer die darin enthaltenen Informationen sowie die Signatur, sodass dieser eine fundierte Entscheidung über die Vertrauenswürdigkeit des digitalen Inhalts treffen kann.

1.3.3. Designziele

Bei der Erstellung der C2PA-Architektur war es wichtig, einige klare Ziele für die Arbeit festzulegen, um sicherzustellen, dass die Technologie weltweit für ein breites Spektrum von Hardware- und Software-Implementierungen nutzbar und für alle zugänglich ist. Diese Ziele sind in Tabelle 1, „C2PA-Designziele", aufgeführt.

Tabelle 1. C2PA-Designziele

Ziel	Beschreibung
Datenschutz	Benutzern die Kontrolle über den Datenschutz ihrer Informationen ermöglichen, einschließlich Verbrauchsdaten und in der Herkunftsquelle aufgezeichneten Informationen
Verantwortung	Sicherstellen, dass Verbraucher die Herkunft eines Vermögenswerts bestimmen können
Skalierbarkeit	Ermöglichen Sie die Erstellung/Nutzung/Validierung der Medienherkunft im gleichen Umfang wie die Erstellung/Nutzung von Medien im Internet.
Erweiterbarkeit	Sicherstellen, dass zukünftige Metadaten- und Berechtigungsanbieter ihre Informationen hinzufügen können, ohne dass eine Eingabe oder Genehmigung durch die C2PA erforderlich ist
Interoperabilität	Sicherstellen, dass unterschiedliche Implementierungen ohne Mehrdeutigkeiten miteinander kompatibel sind
Anwendbarkeit auf den gesamten Workflow	Wahren Sie die Herkunft der Ressource über mehrere Tools hinweg, von der Erstellung über alle nachfolgenden Änderungen bis hin zur Veröffentlichung/Verbreitung.
Technologischer Minimalismus	Erstellen Sie nur die minimal erforderliche neue Technologie in der Spezifikation, indem Sie sich auf bereits etablierte Techniken stützen.
Sicherheit	Entwerfen Sie das System so, dass Verbraucher der Integrität und Herkunft vertrauen können, und lassen Sie das Design von Experten überprüfen.
Allgegenwärtigkeit von Inhalten	Ermöglichen Sie die Angabe der Herkunft für alle gängigen Medientypen, einschließlich Dokumenten.
Flexible Lokalität	Ermöglichen Sie sowohl die Online- als auch die Offline-Speicherung (nur Assets) und die Nutzung/Validierung der Herkunft.
Globale Universalität	Design für die Bedürfnisse interessierter Nutzer weltweit
Barrierefreiheit	Stellen Sie sicher, dass die Technologie in einer Weise genutzt werden kann, die anerkannten Standards für Barrierefreiheit entspricht, wie z. B. WCAG
Schäden und Missbrauch	Entwerfen Sie das Design so, dass potenzielle Schäden, einschließlich Bedrohungen der Menschenrechte und unverhältnismäßiger Risiken für schutzbedürftige Gruppen, vermieden und gemindert werden
Weiterentwicklung	Kontinuierliche Überprüfung der Spezifikationen anhand dieser Ziele, um sicherzustellen, dass sie unsere Priorität bleiben

Kapitel 2. Glossar

2.1. Einführende Begriffe

2.1.1. Akteur

Eine Person oder ein nicht-menschliches Element (Hardware oder Software), das am C2PA-Ökosystem teilnimmt. Beispiele: eine Kamera (Aufnahmegerät), Bildbearbeitungssoftware, Cloud-Dienst oder die Person, die solche Tools verwendet.

HINWEIS Eine Organisation oder Gruppe von Akteuren kann ebenfalls als Akteur im C2PA-Ökosystem betrachtet werden.

2.1.2. Anspruchsgenerator

Der nicht-menschliche (Hardware- oder Software-)Akteur, der die Forderung zu einem Vermögenswert sowie die Forderungssignatur generiert und somit zum zugehörigen C2PA-Manifest des Vermögenswerts führt.

2.1.3. Unterzeichner

Der Inhaber einer Berechtigung für einen privaten Schlüssel, der zum Signieren des Anspruchs verwendet wird. Der Unterzeichner wird durch den Gegenstand der Berechtigung identifiziert.

2.1.4. Manifest-Verbraucher

Ein Akteur, der eine Ressource mit einem zugehörigen C2PA-Manifest verbraucht, um die Herkunftsdaten aus dem C2PA-Manifest zu erhalten.

2.1.5. Validator

Ein Manifest-Verbraucher, dessen Aufgabe es ist, die in der Validierung beschriebenen Aktionen durchzuführen.

2.1.6. Aktion

Eine von einem Akteur an einem Asset durchgeführte Operation. Zum Beispiel „Erstellen“, „Einbetten“ oder „Filter anwenden“.

2.2. Assets und Inhalte

2.2.1. Digitale Inhalte

Der Teil eines Assets, der den tatsächlichen Inhalt darstellt, wie beispielsweise die Pixel eines Bildes, zusammen mit allen zusätzlichen technischen Metadaten, die zum Verständnis des Inhalts erforderlich sind (z. B. ein Farbprofil oder Codierungsparameter).

2.2.2. Asset-Metadaten

Nicht-technische Informationen über das Asset und seinen digitalen Inhalt.

2.2.3. Asset

Eine Datei oder ein Datenstrom, der digitale Inhalte, Asset-Metadaten und optional ein C2PA-Manifest enthält.

HINWEIS	Für die Zwecke dieser Definition erweitern wir die typische Definition von „Datei“ um cloudnative und dynamisch generierte Daten.

2.2.4. Abgeleitete Ressource

Ein abgeleiteter Vermögenswert ist ein Vermögenswert, der aus einem bestehenden Vermögenswert erstellt wird, indem Maßnahmen durchgeführt werden, die dessen digitalen Inhalt verändern.

BEISPIEL: Ein gekürzter Audiostream oder ein Dokument, dem Seiten hinzugefügt wurden.

2.2.5. Asset-Wiedergabe

Eine Darstellung eines Assets (entweder als Teil eines Assets oder als völlig neues Asset), bei der der digitale Inhalt einer „nicht redaktionellen Transformation” (z. B. Neucodierung oder Skalierung) unterzogen wurde.

BEISPIEL: Eine Videodatei, die für eine geringere Bildschirmauflösung oder Netzwerkbandbreite neu codiert wurde.

2.2.6. Zusammengesetztes Asset

Ein zusammengesetztes Asset ist ein Asset, das durch Zusammenstellen mehrerer Teile oder Fragmente digitaler Inhalte (als „Bestandteile” bezeichnet) aus einem oder mehreren anderen Assets erstellt wird. Wenn es auf einem vorhandenen Asset basiert, handelt es sich um einen Sonderfall eines abgeleiteten Assets – ein zusammengesetztes Asset kann jedoch auch von Grund auf neu erstellt werden.

BEISPIELE:

- Ein Video, das durch Importieren vorhandener Videoclips und Audiosegmente in eine „leere Leinwand“ erstellt wurde.
- Ein Bild, bei dem ein anderes Bild importiert und über das Ausgangsbild gelegt wird.

2.2.7. Redaktionelle Umgestaltung

Eine Art der Transformation, die entweder die Absicht oder die Bedeutung oder beides des digitalen Inhalts verändert.

2.3. Kernaspekte von C2PA

2.3.1. Assertion

Eine Datenstruktur, die eine Aussage darstellt, die entweder vom Unterzeichner gemacht (oder „erstellt") oder einfach zum Zeitpunkt der Anspruchserstellung in Bezug auf das Asset gesammelt wurde. Diese Daten sind Teil des C2PA-Manifests.

2.3.2. Anspruch

Eine digital signierte und manipulationssichere Datenstruktur, die auf eine Reihe von Aussagen zu einem Vermögenswert und die zur Darstellung der verbindlichen Inhalte erforderlichen Informationen verweist. Wurden Aussagen redigiert, wird eine entsprechende Erklärung beigefügt. Diese Daten sind Teil des C2PA-Manifests.

2.3.3. Anspruchsignatur

Die digitale Signatur auf dem Anspruch, die mit dem privaten Schlüssel des Unterzeichners erstellt wurde. Die Anspruchsignatur ist Teil des C2PA-Manifests.

2.3.4. C2PA-Manifest

Die Gesamtheit der Informationen über die *Herkunft* eines Assets, basierend auf der Kombination einer oder mehrerer Aussagen (einschließlich Inhaltsbindungen), einer einzelnen Anspruch und einer Anspruchsignatur. Ein C2PA-Manifest ist Teil eines C2PA-Manifest-Speichers.

HINWEIS Ein C2PA-Manifest kann auf andere C2PA-Manifeste verweisen.

2.3.5. C2PA-Manifest-Speicher

Eine Sammlung von C2PA-Manifesten, die entweder in ein Asset eingebettet oder außerhalb des Assets gespeichert werden können.

2.3.6. Inhaltszertifikat

Dies ist der bevorzugte nicht-technische Begriff für ein C2PA-Manifest. Der C2PA-Manifest-Speicher repräsentiert daher die Inhaltsberechtigungsnachweise eines Assets.

Content Credentials bezieht sich auch auf die gesamte C2PA-Technologie und wird daher im Wesentlichen als Plural behandelt. Wenn ein C2PA-Manifest ein Content Credential ist, dann sind mehrere C2PA-Manifeste oder das umfassendere, universelle Konzept Content Credentials.

2.3.7. Aktives Manifest

Das letzte Manifest in der Liste der C2PA-Manifeste innerhalb eines C2PA-Manifest-Speichers, das über die Reihe von Inhaltsbindungen verfügt, die validiert werden können.

2.3.8. Herkunft

Das logische Konzept zum Verständnis der Geschichte eines Assets und seiner Interaktion mit Akteuren und anderen Assets, dargestellt durch die Herkunftsdaten.

2.3.9. Herkunftsdaten

Die Reihe von C2PA-Manifesten für einen Vermögenswert und, im Falle eines zusammengesetzten Vermögenswerts, dessen Bestandteile.

HINWEIS Ein C2PA-Manifest kann auf andere C2PA-Manifeste verweisen.

2.3.10. Authentizität

Eine Eigenschaft digitaler Inhalte, die eine Reihe von Fakten umfasst (z. B. Herkunftsdaten und Hard Bindings), die kryptografisch als unverfälscht verifiziert werden können.

2.3.11. Inhaltsbindung

Informationen, die digitale Inhalte mit einem bestimmten C2PA-Manifest verknüpfen, das mit einem bestimmten Asset verbunden ist, entweder als Hard Binding oder als Soft Binding.

2.3.12. Hard Binding

Ein oder mehrere kryptografische Hashes, die entweder das gesamte Asset oder einen Teil davon eindeutig identifizieren.

2.3.13. Weiche Bindung

Eine Inhaltskennung, die entweder (a) statistisch nicht eindeutig ist, wie beispielsweise ein Fingerabdruck, oder (b) als unsichtbares Wasserzeichen in den identifizierten digitalen Inhalten eingebettet ist.

2.3.14. Vertrauenssignale

Die Sammlung von Informationen, die das Urteil eines Manifest-Verbrauchers über die Vertrauenswürdigkeit eines Assets beeinflussen können. Diese kommen zu dem Unterzeichner hinzu, auf den sich das grundlegende Vertrauensmodell stützt.

2.3.15. C2PA-Vertrauensliste

Eine von C2PA verwaltete Liste von X.509-Zertifikatsvertrauensankern, die Zertifikate an Hardware- und Software-Signaturnutzer ausstellen, die diese zum Signieren von Ansprüchen verwenden.

2.4. Zusätzliche Bedingungen

2.4.1. Dauerhafte Inhaltsberechtigungsnachweis

Eine dauerhafte Inhaltsberechtigungsnachweis ist ein Inhaltsberechtigungsnachweis, für den eine oder mehrere Soft-Bindungen existieren, die seine Auffindung in einem Manifest-Repository ermöglichen.

2.4.2. Fingerabdruck

Eine Reihe von inhärenten Eigenschaften, die aus digitalen Inhalten berechnet werden können und die den Inhalt oder nahezu identische Duplikate davon identifizieren.

> **BEISPIEL:** Ein Asset kann aufgrund der Entfernung oder Beschädigung von Asset-Metadaten von seinem C2PA-Manifest getrennt werden. Ein Fingerabdruck des digitalen Inhalts des Assets könnte verwendet werden, um eine Datenbank zu durchsuchen und das Asset mit einem intakten C2PA-Manifest wiederherzustellen.

2.4.3. Unsichtbares Wasserzeichen

Informationen, die auf eine für den Menschen im Wesentlichen nicht wahrnehmbare Weise in den digitalen Inhalt eines Vermögenswerts eingebettet sind und beispielsweise zur eindeutigen Identifizierung des Vermögenswerts oder zur Speicherung eines Verweises auf ein C2PA-Manifest verwendet werden können.

2.4.4. Sichtbares Wasserzeichen

Ein wahrnehmbarer Bestandteil des digitalen Inhalts, der für Menschen lesbare Informationen über die Herkunft des Vermögenswerts enthält.

2.4.5. Manifest-Repository

Ein Repository, in dem C2PA-Manifeste und C2PA-Manifest-Speicher abgelegt werden können und das mithilfe einer Inhaltsbindung durchsucht werden kann.

2.5. Übersicht

Dieses Bild zeigt, wie all diese verschiedenen Elemente zusammenkommen, um die C2PA-Architektur darzustellen.

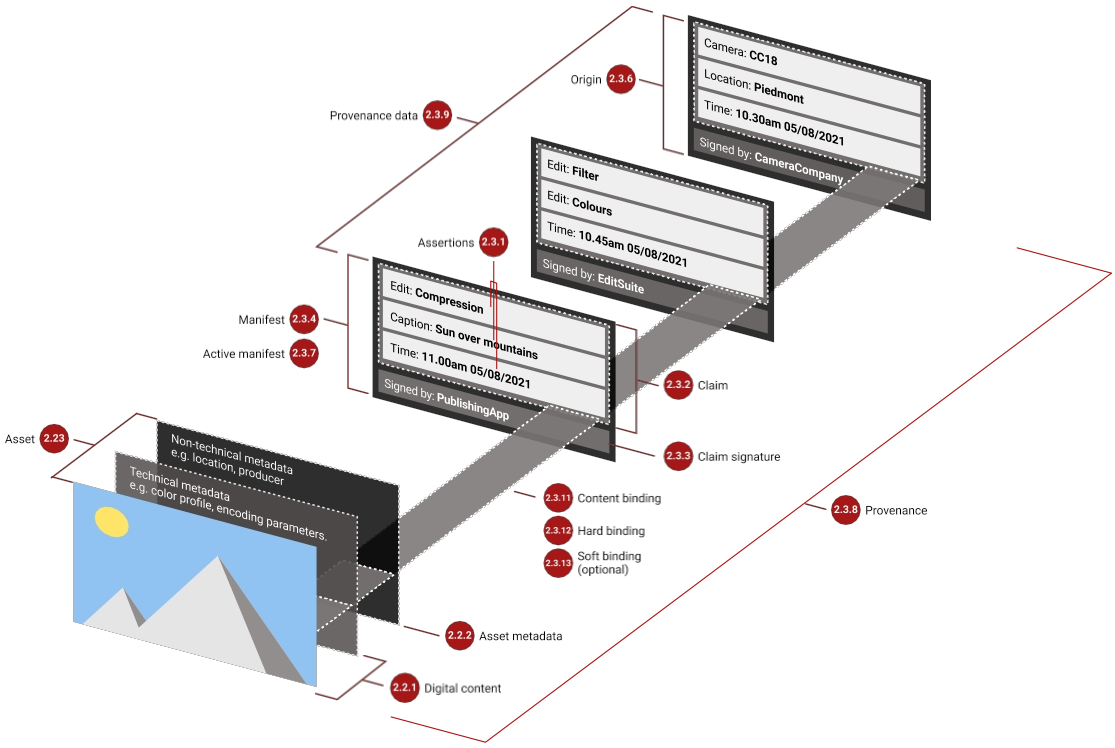

Abbildung 3. Elemente von C2PA

Kapitel 3. Normative Verweise

3.1. Kernformate

- CBOR
- JSON
- JSON-LD
- JPEG Universal Metadata Box Format (JUMBF)

3.2. Schemas

- CDDL
- JSON-Schema
- Dublin Core Metadata Initiative

3.3. Digitale und elektronische Signaturen

- Kryptografische Nachrichtensyntax (CMS)
- Internet X.509 PKI Zeitstempelprotokoll
- Algorithmen und Identifikatoren für die Internet X.509 Public Key Infrastructure Certificate und Certificate Revocation List (CRL) Profile
- Internet X.509 Public Key Infrastructure: Zusätzliche Algorithmen und Identifikatoren für DSA und ECDSA
- Sichere Hash-Algorithmen der USA
- Online Certificate Status Protocol (OCSP)
- JSON-Webalgorithmen (JWA)
- PKCS #1: RSA-Kryptografie-Spezifikationen Version 2.2
- Edwards-Kurven-Algorithmus für digitale Signaturen (EdDSA)
- CBOR-Objektsignierung und -verschlüsselung (COSE)
- Verwendung von RSA-Algorithmen mit COSE-Nachrichten
- Algorithmus-Identifikatoren für Ed25519, Ed448, X25519 und X448 zur Verwendung in der Internet-X.509-Public-Key-Infrastruktur
- X.509-Zertifikat – Erweiterte Schlüsselverwendung (EKU) für allgemeine Zwecke zur Dokumentensignatur
- CBOR-Objektunterzeichnung und -verschlüsselung (COSE): Header-Parameter für die Übertragung und Referenzierung von X.509-Zertifikaten
- Internet X.509 Public Key Infrastructure: Logos in X.509-Zertifikaten
- JSON Advanced Electronic Signatures (JAdES)

3.4. Einbettbare Formate

- ISO-Basis-Medien-Dateiformat (BMFF)
- PDF 1.7
- PDF 2.0
- JPEG 1
- JPEG XT, ISO/IEC 18477-3
- JPEG XL, ISO/IEC 18181-2:2024
- PNG
- SVG
- GIF
- ID3
- Digital Negative oder DNG
- TIFF/EP
- TIFF v6)
- RIFF
- Multi-Picture Format (MPF)
- Offenes Schriftformat
- OpenType

3.5. Sonstiges

- eXtensible Metadata Platform (XMP)
- JSON-LD-Serialisierung von XMP
- IPTC-Standard für Foto-Metadaten
- Exif
- UUID
- Uniform Resource Names (URNs)
- Universell eindeutige Identifikatoren (UUIDs)
- ISO 8601
- RFC 3339
- RFC 2326
- Medienfragmente

- Web-Annotations-Datenmodell
- Brotli-komprimiertes Datenformat
- RFC 5646, BCP 47

Kapitel 4. Standardbegriffe

Die Schlüsselwörter „MUSS“, „DARF NICHT“, „ERFORDERLICH“, „SOLL“, „SOLL NICHT“, „SOLLTE“, „SOLLTE NICHT“,
Die Begriffe „EMPFOHLEN“, „NICHT EMPFOHLEN“, „KANN“ und „OPTIONAL“ in diesem Dokument sind gemäß BCP 14, RFC 2119 und RFC 8174 zu interpretieren, unabhängig davon, in welcher Schreibweise (Großbuchstaben, Kleinbuchstaben oder gemischt) sie erscheinen.

Kapitel 5. Versionierung

5.1. Kompatibilität

Mit der Weiterentwicklung der Content Credentials-Spezifikation haben sich auch Konstrukte wie Box-Labels, Assertions (und ihre Felder), Claims und Zeitstempel weiterentwickelt. Es wurden neue Assertions hinzugefügt, und einige bestehende Assertions und Claims haben neuere Versionen mit zusätzlichen Feldern. Darüber hinaus wurden einige Konstrukte als veraltet markiert. Wenn in dieser Spezifikation ein Konstrukt als veraltet markiert ist, bedeutet dies, dass ein Claim-Generator dieses Konstrukt (oder diesen Wert) nicht schreiben soll, aber dass ein Validator es lesen sollte.

Um die Interoperabilität zwischen Anspruchgeneratoren und Validatoren zu erleichtern, gibt ein Anspruchgenerator an, welche Version der Spezifikation er zur Generierung des Anspruchs verwendet. Wenn ein Anspruchgenerator angibt, dass er eine bestimmte Version der Spezifikation verwendet, erklärt er damit, dass das aktive Manifest des Assets in Übereinstimmung mit dieser Version der Spezifikation erstellt wurde und somit keine veralteten Konstrukte enthält, die unter dieser Version der Spezifikation in Tabelle 19, „Status der Konstrukte”, in Anhang C, *„Überlegungen zur Veraltung”,* aufgeführt sind.

HINWEIS

Diese Spezifikation schreibt keine bestimmte technische Vorgehensweise für diese Erklärung vor, es wird jedoch erwartet, dass durch andere Mittel eine Anleitung bereitgestellt wird.

Ein Validator muss mit mindestens einer Version der Spezifikation kompatibel sein, kann jedoch auch mit weiteren Versionen kompatibel sein. Ein Validator, der mit einer bestimmten Version der Spezifikation kompatibel ist, muss alle für diese Version aufgeführten nicht veralteten Konstrukte unterstützen. Wenn der Validator auf ein Manifest stößt, das Konstrukte aus einer Version der Spezifikation verwendet, die der Validator nicht unterstützt (entweder weil sie veraltet oder unbekannt sind), kann er das veraltete Konstrukt ignorieren und den Rest des Manifests so verarbeiten, als ob dieses Konstrukt nicht vorhanden wäre. Alternativ kann der Validator das gesamte Manifest als unbekannten Ursprungs behandeln, indem er je nach Fall den Statuscode `ingredient.unknownProvenance` oder `manifest.unknownProvenance` zurückgibt.

5.2. Versionshistorie

5.2.1. 2.2 – Mai 2025

Diese Version konzentriert sich sowohl auf technische als auch auf redaktionelle Änderungen an der Spezifikation, um einige der neuen Funktionen von 2.1 zu verdeutlichen und gleichzeitig den Anforderungen der Implementierer gerecht zu werden. Die Spezifikation wurde aktualisiert, um den neuesten Best Practices in diesem Bereich Rechnung zu tragen.

- Neue ergänzende Spezifikationen für die Soft Binding Resolution API hinzugefügt
- Neue Felder zur Angabe der Wiederherstellung von Soft-Binding-Manifesten zur Inhaltsangabe hinzugefügt
- Unterstützung für mehrteilige Assets wie Android Motion Photos hinzugefügt
- Unterstützung für das Hinzufügen von Zeitstempeln und Widerrufsinformationen in einem Update-Manifest hinzugefügt, wodurch Zeitstempel-Manifeste ersetzt werden
- Unterstützung für eine „beanspruchte Signaturerstellungszeit” hinzugefügt

- Unterstützung für neue `c2pa-kp-claimSigning` EKU hinzugefügt
- Eingeschränkte Verwendung der C2PA-Vertrauensliste für Zertifikate mit der EKU „`c2pa-kp-claimSigning`"
- Eingeführt `digitalSourceType`-Werte `http://c2pa.org/digitalsourcetype/trainedAlgorithmicData` (ersetzt `c2pa.trainedAlgorithmicData`) und `http://c2pa.org/digitalsourcetype/empty`
- Datenfelder durch eingebettete Datenaussagen ersetzt
- Zusätzliche Hinweise zum Zurücksetzen redigierter Assertions bereitgestellt
- Verwendung von `created_assertions` und `gathered_assertions` in Bezug auf das Vertrauensmodell präzisiert
- Terminologie rund um „Unterzeichner" und „Anspruchsgenerator" in Bezug auf ihre Rollen präzisiert
- Änderungen und Verbesserungen bei verschiedenen festen Bindungsaussagen
    - Erlaubt `c2pa.hash.data`, klassische Metadatenabschnitte eines Assets auszuschließen
    - Unterstützung für Ausschlüsse in der `c2pa.hash.boxes`-Assertion hinzufügen
    - Unterstützung für die Verwendung einer `c2pa.hash.bmff`-Assertion in einem Update-Manifest hinzufügen
- Klarstellung, welche JUMBF-Boxen im Assertion Store zulässig sind
- Klärung der Handhabung von Zertifikatswiderrufen
- Klarstellung zur Zeitstempelvalidierung
- Verbesserungen und Klarstellungen bei Aktionsassertionen
- Verbesserungen an Soft-Binding-Assertions
- Überarbeitung der BMFF-Hashing-Diagramme zur Verbesserung der Klarheit und Korrektheit
- Die Anforderung, dass alle Manifeste in einem Manifest-Speicher referenziert werden müssen, wurde entfernt.

5.2.2. 2.1 – September 2024

Diese Version konzentriert sich auf technische und redaktionelle Änderungen an der Spezifikation, um die Sicherheit und Zuverlässigkeit von Content Credentials zu verbessern. Alle öffentlich bekannt gewordenen Sicherheitslücken wurden behoben, und die Spezifikation wurde aktualisiert, um den neuesten Best Practices in diesem Bereich Rechnung zu tragen.

- Klare Definitionen der Manifest- und Asset-Zustände
    - Wohlgeformte Manifeste
    - Gültige Manifeste
    - Vertrauenswürdige Manifeste
    - Gültige Assets
- Klare Definitionen und Prozesse für den Umgang mit Veralterung und Versionierung
- Neuer `c2pa`-URN-Namensraum für die Kennzeichnung von Manifesten!

- einschließlich einer vollständig spezifizierten ABNF

        - Neue `Zutaten v3`-Aussage
    - Unterstützt umfangreichere Modelle von Workflows auf Basis von Inhaltsstoffen.
    - Unterstützung für `dataTypes` und `claimSignature`.
    - Felder wurden umbenannt, um eine größere Übereinstimmung mit anderen Assertions zu erreichen.
    - Neue Validierungsstatusfelder wurden hinzugefügt, um die neuen Statusinformationen zu ergänzen
    - `dc:title` und `dc:format` sind jetzt optional
- Neue `c2pa.hash.bmff.v3`-Assertion
    - Unterstützt das Hashing von festen und variablen Blockgrößen für BMFF-basierte Assets.
- Neues Zeitstempel-Manifest
    - Festlegung einer „Existenzzeit“ für einen bestimmten Vermögenswert.
    - Ähnlich wie bei einem Aktualisierungsmanifest, jedoch mit einem TSA als Unterzeichner
- Verbessertes Modell für die Durchführung von Standard-Zeitstempeln gemäß RFC 3161.
    - `sigTst2` & CTT-Zeitstempel
    - Einführung der neuen C2PA TSA-Vertrauensliste
- Verbesserungen bei der Validierung
    - Detaillierte Validierungsanweisungen für alle Standardaussagen
    - Bei Verwendung der `Zutaten`-Assertion ist nun eine Validierung der Zutaten erforderlich
    - Erweiterte Inhaltsstoffvalidierung für detailliertere Statusinformationen
    - Unterstützung für die Validierung von redigierten Angaben zu Inhaltsstoffen
    - Hinzufügung detaillierter Anforderungen für die Validierung von Zeitstempeln
    - Hash-URIs zu Datenfeldern und allen benutzerdefinierten Feldern werden nun validiert
    - Definiertes Verfahren für den Umgang mit Manifesten mit übereinstimmenden eindeutigen IDs
    - Behandlung von „verwaisten Manifesten” im Validierungsprozess
    - Zahlreiche neue Validierungsstatuscodes, darunter ein neuer Code-Typ „informativ”
- Verbesserungen bei der Dokumentation und Sicherheit von Hash-Verfahren
    - BMFF-basierte Assets
    - „Allgemeine Boxen”
    - ZIP
- Der Abschnitt zur Format-Einbettung wurde in einen eigenen Anhang verschoben.
    - Unterstützung für JPEG-XL hinzugefügt

- Verbesserungen bei Soft-Bindings
- Verbesserungen bei Aktionsaussagen
    - Entweder `c2pa.created` oder `c2pa.opened` ist nun in einem Standard-Manifest obligatorisch
    - Es wurden einige neue Standardaktionstypen hinzugefügt.
    - Es ist nun möglich, mehrere Aktionsaussagen in einem einzigen Manifest zu haben
    - Aktionsvorlagen werden nun anhand weiterer Beispiele besser erläutert.
    - RFC 3339-basierte Regionen von Interesse
- Die verschiedenen Arten/Formen eindeutiger Identifikatoren für Assets wurden präzisiert.
- Fehlende Kompatibilitätsunterstützung für JPEG hinzugefügt Vertrauen
- Alle CDDLs wurden bereinigt, einschließlich der Entfernung normativer Formulierungen.
- Und verschiedene redaktionelle Verbesserungen
    - Benutzerdefinierte Bezeichnungen wurden zu einem benutzerdefinierten Benennungsschema umdefiniert.
    - Einbettung in PDFs
    - Zahlreiche redaktionelle Verbesserungen zur Vorbereitung des Dokuments für die Standardisierung durch ISO

5.2.3. 2.0 – Januar 2024

Diese Version stellt eine erhebliche Abweichung von früheren Versionen dar. Der Begriff „Akteur“ wird weniger häufig verwendet und bezieht sich nicht mehr auf Menschen und Organisationen. Zusätzlich zu den vom Validierer konfigurierten Vertrauenslisten wird eine neue Standard-Vertrauensliste eingeführt, die „C2PA-Vertrauensliste“, die für Zertifikate gelten soll, die für Hardware und Software ausgestellt wurden. Diese philosophische Änderung führte zu folgenden funktionalen Änderungen in der Spezifikation:

- Für die Signatur dürfen nur X.509-Zertifikate verwendet werden.
- Verbesserungen in den Abschnitten „Validierung” und „Vertrauensmodell”
    - Einführung der Konzepte „wohlgeformte“ und „gültige“ C2PA-Manifeste
    - Klärt verschiedene Aspekte des Validierungsprozesses
- Verfeinerung der Metadatenverarbeitung
    - Die veralteten Metadaten-Assertions Exif, IPTC und Schema.org wurden entfernt.
    - ein neues allgemeines Konzept für „Metadaten-Aussagen” definiert
    - `c2pa.metadata` erlaubt nur einen festen Satz von Schemata und Werten
    - Der Prozess zur Erstellung von `c2pa.metadata` ist nun detaillierter dokumentiert
    - Die Abschnitte zur XMP-Verarbeitung wurden überarbeitet, um relevante Änderungen widerzuspiegeln
    - Die Empfehlungen zum Hashing von Standard-Metadaten-Speicherorten außerhalb des Manifests wurden verbessert.
- Der Abschnitt „W3C Verifiable Credentials” wurde entfernt.

    - Alle Verweise darauf und auf den VC Store wurden entfernt.
    - Das Feld „Akteure" wurde aus der Aktionsaussage entfernt.
    - Identifizierte Personen aus den Metadaten der Behauptung entfernt
- Die Assertion „Training & Data Mining" wurde entfernt
- Die Assertion „Empfehlungen" wurde entfernt.

Darüber hinaus wurden die folgenden weiteren Änderungen vorgenommen, um verschiedene Aspekte der Spezifikation zu verbessern:

- Version v2 der Behauptung.
    - Entfernt veraltete und ungenutzte Felder
    - Aufteilung der Assertions in created_assertions und gathered_assertions
    - Es ist nur ein einziger Anspruchgenerator zulässig, bei dem es sich um den Unterzeichner handeln muss
    - claim-generator-info verfügt nun über ein spezifisches Feld operating_system
- Box-basiertes Hashing wird nun dringend für alle Formate empfohlen, die dies unterstützen.
- Die veraltete c2pa.hash.bmff-Assertion wurde entfernt.
- Neue Aktion „c2pa.watermarked" hinzugefügt
- c2pa.font-Aktionen sind jetzt nur noch Schriftartenaktionen
    - Auch c2pa.font.info ist nun nur noch font.info
- Die Darstellung von CDDL-Schemas wurde bereinigt.
- Einige normative Verweise wurden aktualisiert und Hinweise zu zukünftigen Versionen entfernt
- Zahlreiche redaktionelle Verbesserungen, darunter korrigierte Links

5.2.4. 1.4 – November 2023

- Unterstützung für die Einbettung eines C2PA-Manifests in ein ZIP-basiertes Format (z. B. EPUB, OOXML, ODF, OpenXPS) hinzugefügt
- Manifeste können nun in einer speziellen Brob-Box komprimiert werden.
- Unterstützung für mehrere Dateien, auch bekannt als Sammlung, Hashing hinzugefügt
- Neue Bereiche von Interesse für textbasierte Formate (z. B. PDF, Office, EPUB usw.) hinzugefügt
- Neue c2pa.metadata-Assertion zur Unterstützung von Exif, IPTC, Schema.org und XMP hinzugefügt
- Umfassende Überarbeitung der Unterstützung für TIFF-Einbettungen
- Unterstützung für die Einbettung von C2PA-Manifesten in OpenType- und TrueType-Schriftarten hinzugefügt
- Unterstützung für Manifeste auf Objektebene in PDF eingeführt
- Erweiterte Unterstützung für Link-Header für eingebettete Manifeste
- Klärung von Problemen mit Box-Hashing

- Klärung von Problemen bei der Signierung, einschließlich Zeitstempel, PKIStatus und Dokumentensignierung EKU
- Anpassung an Exif 3.0
- Verbesserungen an den CDDL-Schemas
- Zahlreiche redaktionelle Verbesserungen

5.2.5. 1.3 – April 2023

- Neue Version v2 der Aktionsaussage mit Unterstützung für viele neue Optionen
- Neue Version v2 der Inhaltsstoffangabe mit Unterstützung für eingebettete Daten
- Neue Asset-Referenz- und Asset-Typ-Assertions
- Neue Datenfelder zum Speichern beliebiger Daten im Manifest
- Neue allgemeine Box-Hash-Methodik für ein umfassenderes Byte-Bereich-Hashing
- Neue Datenstrukturen für „Regions of Interest“, die auf verschiedene Assertions angewendet werden können
- Hinzufügen der Dokumentensignatur-EKU als alternative Standard-EKU für C2PA-Signaturnutzer, wenn ein Validator nicht mit einer EKU-Liste konfiguriert ist
- Neues Feld `„digitalSourceType“` für die Verwendung durch C2PA hinzugefügt
- Unterstützung für viele neue Formate hinzugefügt: MPF, WebP, AIFF, AVI, GIF
- Aktualisiertes Entitätsdiagramm, um Ergänzungen seit 1.0 widerzuspiegeln
- Aktualisierung der COSE-Header-Definition für X.509-Zertifikate gemäß RFC 9360
- Aktualisierung der Anleitung zum Einbetten von PDF-Dateien und deren Zusammenhang mit PDF-Signaturen
- Aktualisierte Informationen zu JUMBF-Hashing und JUMBF-Box-Umschaltern
- Veraltete Version 1 des BMFF-Hash
- Verwendung der JUMBF-Schutzbox in einem C2PA-Manifest präzisiert
- Klarstellung der C2PA-spezifischen Anforderung, dass alle Zwischenzertifikate nach X.509 in COSE-Signaturen enthalten sein müssen
- Es wurde klargestellt, dass Zeitstempel unbegrenzt gültig sind.
- Zahlreiche redaktionelle Verbesserungen!

5.2.6. 1.2 – Oktober 2022

- Details zum Einbetten eines C2PA-Manifests in DNG oder TIFF hinzugefügt
- Neues Feld `„digitalSourceType“` zu „Actions” hinzugefügt
- Änderung von `„stds.iptc.photometadata“` in `„stds.iptc“,` um IPTC-Videometadaten zu unterstützen
- Klarstellung der Versionierung von Assertions beim Hinzufügen optionaler Felder

5.2.7. 1.1 – September 2022

- Mechanismus zur Unterstützung von Salting-Box-Hashing definiert
- Neue `c2pa.hash.bmff.v2`-Assertion mit Änderungen am Hash-Modell zur Verbesserung der Sicherheit
- Aktivieren Sie die Assertion-Metadaten für den Anspruch
- Ersetzen von `claim_generator_hints` durch `claim_generator_info`
- Neue Assertion hinzugefügt, um das Konzept der Endorsements zu unterstützen
- Verbesserungen an der `c2pa.actions`-Assertion
- Alle Fehler- und Statuscodes haben nun das Präfix `c2pa`
- Mechanismus für die Redaktion von W3C-VCs definiert
- Klärung der Validierung von EKUs in Zertifikaten
- Überarbeitung des Validierungsalgorithmus zur Berücksichtigung technischer Änderungen
- Korrekturen an den CDDL- und JSON-Schemas zur Anpassung an den normativen Text
- Überarbeitung der Abbildungen zur Berücksichtigung der Änderungen
- Verschiedene redaktionelle und typografische Korrekturen
- Aktualisierung normativer Verweise (einschließlich JUMBF & W3C VC-Datenmodell)

5.2.8. 1.0 – Dezember 2021

- Erstveröffentlichung

Kapitel 6. Behauptungen

6.1. Allgemeines

Es wird erwartet, dass jeder Anspruchgenerator, der von Akteuren im System verwendet wird, das einen Vermögenswert erstellt oder verarbeitet, eine oder mehrere Aussagen darüber erstellt oder zusammenstellt, wann, wo und wie der Vermögenswert entstanden ist oder umgewandelt wurde. Eine Aussage ist gekennzeichnete Daten, typischerweise (wenn auch nicht zwingend) in einer CBOR-basierten Struktur, die eine Erklärung zu einem Asset darstellt. Einige dieser Aussagen enthalten von Menschen generierte Informationen (z. B. Alternativtext für Barrierefreiheit), während andere von Maschinen (Software/Hardware) stammen, die die von ihnen generierten Informationen bereitstellen (z. B. Kameratyp).

Beispiele für Aussagen sind:

- Metadaten (z. B. Kamerainformationen wie Hersteller oder Objektiv);
- auf das Asset angewendete Aktionen (z. B. Zuschneiden, Farbkorrektur);
- Miniaturansicht des Assets oder seiner Bestandteile;
- Inhaltsbindungen (z. B. kryptografische Hashes).

Bestimmte Aussagen können durch nachfolgende Ansprüche redigiert werden (siehe Abschnitt 6.8, „Redigieren von Aussagen”), aber sie können nicht mehr geändert werden, sobald sie als Teil eines Anspruchs gemacht wurden.

6.2. Labels

6.2.1. Namensräume

Zeichenfolgenwerte in C2PA-Datenstrukturen können unter Verwendung eines Punktes (`.`) als Trennzeichen in Namensräumen organisiert werden. Der C2PA-Namensraum „`c2pa`“ muss am Anfang jedes in dieser Spezifikation definierten Zeichenfolgenwerts stehen. Entitätsspezifische Namensräume müssen mit dem Internet-Domänennamen für die Entität beginnen, ähnlich wie Java-Pakete definiert sind (z. B. `com.litware`, `net.fineartschool`).

Die durch Punkte getrennten Komponenten eines entitätsspezifischen Namensraums müssen der in POSIX oder C Locale festgelegten Konvention für die Benennung von Variablen (`[a-zA-Z0-9][a-zA-Z0-9_-]*`) entsprechen, wie in der folgenden ABNF (ABNF für Namensräume) definiert.

ABNF für Namespaces

```
qualifizierter-Namespace = „c2pa” / Entität
entity = entity-component *( „.“ entity-component )
Entitätskomponente = 1( ZIFFER / ALPHA ) *( ZIFFER / ALPHA / „-“ / „_“ )
```

6.2.2. Benennung von Labels

Jede Assertion hat ein Label, das entweder durch die C2PA-Spezifikationen oder eine externe Entität definiert ist. Diese Labels sind Zeichenfolgen, die, wie im vorstehenden Absatz beschrieben, einen Namespace haben oder durch eine Entität definiert sind. Die gängigsten Bezeichnungen werden im c2pa-Namensraum definiert, aber Bezeichnungen können jeden Namensraum verwenden, der den Konventionen entspricht. Bezeichnungen werden auch mit einem einfachen Schema aus aufsteigenden Ganzzahlen versioniert (z. B. `c2pa.actions.v2`). Wenn keine Version angegeben ist, wird sie als `v1` betrachtet. Die Liste der öffentlich bekannten Bezeichnungen finden Sie in Kapitel 18, *C2PA-Standardaussagen.*

HINWEIS Frühere Versionen dieses Dokuments sahen auch Namespaces für etablierte Standards vor. dies wurde jedoch durch die einfache Verwendung von entitätsspezifischen Namespaces (z. B. `org.iso`, `org.w3`) ersetzt.

ABNF für Assertion-Labels

```
namespaced-label = qualified-namespace label
qualified-namespace = „c2pa“ / entity
Entität = Entitätskomponente *( „.“ Entitätskomponente )
Entitätskomponente = 1( ZIFFERE / ALPHA ) *( ZIFFERE / ALPHA / „-“ / „_“ )
Bezeichnung = 1*( „.“ Bezeichnungskomponente )
label-component = 1( DIGIT / ALPHA ) *( DIGIT / ALPHA / "-" / "_" )
```

Die durch Punkte getrennten Komponenten einer Bezeichnung folgen der in POSIX oder C Locale festgelegten Konvention zur Benennung von Variablen (`[a-zA-Z][a-zA-Z0-9_-]*`), mit der Einschränkung, dass die Verwendung eines wiederholten Unterstrichs (___) für die Bezeichnung mehrerer Aussagen desselben Typs reserviert ist.

6.3. Versionierung

Wenn das Schema einer Assertion geändert wird, sollte dies auf rückwärtskompatible Weise erfolgen. Das bedeutet, dass neue Felder hinzugefügt und bestehende Felder als veraltet markiert werden können (d. h. sie können gelesen, aber nicht mehr geschrieben werden). Bestehende Felder dürfen nicht entfernt werden. Die Bezeichnung würde dann aus einer inkrementierten Versionsnummer bestehen, beispielsweise von `c2pa.action` (veraltet) zu `c2pa.action.v2`.

Da das Hinzufügen optionaler Felder unter Beibehaltung der Abwärtskompatibilität möglich ist, können solche Felder zum Schema einer bestehenden Assertion hinzugefügt werden, ohne dass die Versionsnummer geändert werden muss.

Veraltete Felder für C2PA-Standardassertionen sind in Kapitel 18, *C2PA-Standardassertionen*, aufgeführt. Claim-Generatoren dürfen bei der Erstellung von Assertionen keine Daten in veraltete Assertionsfelder einfügen.

In Fällen, in denen eine nicht abwärtskompatible Änderung erforderlich ist, soll die Assertion anstelle einer Erhöhung der Versionsnummer des Labels ein neues Label erhalten.

HINWEIS Beispielsweise könnte `c2pa.ingredient` in das fiktive `c2pa.component` geändert werden.

6.4. Mehrere Instanzen

Mehrere Assertions desselben Typs können im selben Manifest vorkommen, aber da Assertions über Claims über

ihrem Label referenziert werden, müssen die Assertions-Labels eindeutig sein. Dies wird durch Hinzufügen eines doppelten Unterstrichs und eines monoton ansteigenden Index zum Label erreicht. Wenn ein Manifest beispielsweise eine einzelne Assertion vom Typ `c2pa.metadata` enthält, lautet die Assertion-Bezeichnung `c2pa.metadata`. Wenn ein Manifest drei Assertions dieses Typs enthält, lauten die Bezeichnungen `c2pa.metadata`, `c2pa.metadata__1` und `c2pa.metadata__2`.

Wenn eine Kennzeichnung eine Versionsnummer enthält, ist diese Versionsnummer Teil der Kennzeichnung selbst. Bei mehreren Instanzen folgt die Instanznummer daher weiterhin der Kennzeichnung, z. B. `c2pa.ingredient.v2__2`.

6.5. Schema-Validierung

Die in diesem Dokument bereitgestellten Schemata sowie die maschinenlesbaren Schemata, die von der C2PA-Website heruntergeladen werden können, sollten nur als Hilfe zum Verständnis der zu lesenden oder zu schreibenden Syntax verwendet werden. Es ist weder notwendig noch empfehlenswert, dass ein Validierer irgendeine Form der Schema-Validierung durchführt.

6.6. Assertion Store

Die Gesamtheit der Assertions, auf die ein Anspruch in einem Manifest Bezug nimmt, wird in einer logischen Konstruktion zusammengefasst, die als *Assertion Store* bezeichnet wird. Die Assertions und der Assertion Store werden gemäß Abschnitt 11.1, „Verwendung von JUMBF“, gespeichert. Insbesondere muss jede Assertion, auf die in `den created_assertions` oder `gathered_assertions` (jedoch nicht `in den redacted_assertions`) eines Anspruchs verwiesen wird, im Assertion Store vorhanden sein, der sich im selben C2PA-Manifest wie der Anspruch befindet.

In jedem Manifest gibt es einen einzigen Assertion Store. Da jedoch ein Asset mehrere Manifeste haben kann, von denen jedes eine bestimmte Reihe von Assertions darstellt, kann es mehrere Assertion Stores geben, die mit einem Asset verbunden sind.

6.7. Eingebettete vs. extern gespeicherte Daten

Einige Assertionsdaten können aufgrund ihrer Größe oder seltenen Verwendung extern gehostet werden. Solche Daten sind nicht im Assertionsspeicher eingebettet, sondern werden über eine URI referenziert. Dies wird durch eine Cloud-Daten-Assertion erreicht (siehe Abschnitt 18.11, „Cloud-Daten“). Im Gegensatz zu eingebetteten Assertionsdaten werden Cloud-Daten nicht im Rahmen der Manifestvalidierung abgerufen oder validiert, sondern nur dann abgerufen und validiert, wenn sie von einer Anwendung gemäß einem anderen Satz von Validierungsregeln, wie in Abschnitt 15.10, „Validieren der Assertions“, beschrieben, ausdrücklich benötigt werden.

6.8. Redigieren von Assertions

Assertions, die in einem in ein Asset eingebetteten Manifest vorhanden sind, können aus dem Manifest dieses Assets entfernt werden, wenn das Asset als Bestandteil verwendet wird. Dieser Vorgang wird als Redigieren bezeichnet.

Die Redaktion umfasst entweder das Entfernen der gesamten Assertion aus dem Assertion-Speicher des Manifests oder das Beibehalten des gekennzeichneten Assertion-Containers, wobei jedoch die JUMBF-Inhaltsfelder innerhalb dieser Assertion durch ein einziges UUID-Inhaltsfeld ersetzt werden, dessen ID-Feld den Wert `CAA98EEE-9D4D-F80E-86AD-4DFFCA263973` (die sogenannte C2PA-Redaktions-UUID) enthält

und dessen DATA-Feld nur Nullen (binäre 0x00-Werte) enthält.

Darüber hinaus muss dem Anspruch ein Vermerk hinzugefügt werden, dass etwas entfernt wurde, und zwar in Form einer URI-Referenz auf die redigierte Assertion im Feld „redacted_assertions" des Anspruchs. Es wird außerdem dringend empfohlen, dass der Anspruchsgenerator eine c2pa.redacted-Aktionsassertion mit einem redigierten Feld hinzufügt, wie in Abschnitt 18.14.4.7, „Parameter", beschrieben.

Bei der Redigierung einer Inhaltsstoffaussage, die auf ein C2PA-Manifest verweist, muss das zugehörige Manifest aus dem C2PA-Manifest-Speicher entfernt werden, wenn nach der Redigierung keine weiteren Verweise darauf vorhanden sind.

HINWEIS

Da die URI-Referenz jeder Assertion die Assertion-Kennzeichnung enthält, ist auch bekannt, um welche Art von Informationen (z. B. Miniaturansicht, Metadaten usw.) entfernt wurde. Dies ermöglicht es sowohl Menschen als auch Maschinen, Regeln anzuwenden, um zu bestimmen, ob die Entfernung akzeptabel war.

Sofern die Redigierung der Assertion nicht auch eine Änderung des digitalen Inhalts erfordert, ist ein Aktualisierungsmanifest zu verwenden, um die Redigierung zu dokumentieren, da es eine Aussage über die Nichtänderung des Inhalts enthält.

Claim-Generatoren dürfen keine Assertions mit der Bezeichnung c2pa.actions oder c2pa.actions.v2 redigieren, da dieser Assertionstyp wesentliche Informationen zum Verständnis der Historie eines Assets enthält. Sie dürfen auch keine festen Verknüpfungen zu Inhaltsaussagen redigieren – weder c2pa.hash.data, c2pa.hash.boxes, c2pa.hash.collection.data, c2pa.hash.bmff.v2 (veraltet) noch c2pa.hash.bmff.v3, da diese Aussagen für die Bestimmung der Integrität des Assets erforderlich sind.

HINWEIS

Wenn Angaben in einer Zutatenliste redigiert werden, auf die über eine der veralteten Zutatenangaben (c2pa.ingredient oder c2pa.ingredient.v2) verwiesen wird, erfolgt die Validierung von dieser Behauptung fehl (wie in Abschnitt 15.11.3, „Validierung von Inhaltsstoffbehauptungen", beschrieben), da nur c2pa.ingredient.v3-Behauptungen die in Abschnitt 15.11.3.3.1, „Validierungsmethode für Behauptungssignatur-Hash", beschriebene Validierungsmethode für Behauptungssignatur-Hash unterstützen.

6.9. Spezifikationen der Zeit in Angaben

Die Standardspezifikation für einen Datums- und/oder Zeitwert in einer Assertion ist das Datums-/Zeitformat, das in CBOR als Tag-Nummer 0 (RFC 8949, 3.4.1) serialisiert und in CDDL als Typ tdate dargestellt wird.

Es gibt einen Fall, wie beim Hinzufügen einer behaupteten Signaturzeit beschrieben, in dem die Zeit als spezieller Typ von CBOR-Datum/Uhrzeit dargestellt wird.

Darüber hinaus gibt es die Zeitstempel-Assertion, die die im Signaturprozess beschriebenen Standardformate für Zeitstempel verwendet.

Der Grund für die unterschiedlichen Arten der Datums- und Zeitdarstellung besteht darin, für jeden spezifischen Anwendungsfall die am besten geeignete Darstellung auf der Grundlage der bestehenden Standards zu ermöglichen.

Kapitel 7. Datenboxen

WICHTIG	Dieser Abschnitt wird aus historischen Gründen beibehalten. Das Konzept einer Datenbox wurde Veraltet zugunsten einer Standardaussage, die ein Standard-JUMBF-Embedded-File-Content-Type-Feld zur Aufnahme der Daten verwendet. Weitere Informationen finden Sie unter [_data_box].

7.1. Allgemeines

Datenboxen bieten eine Möglichkeit, beliebige Daten in das C2PA-Manifest aufzunehmen, auf das aus einer Assertion verwiesen wird, anstatt sie direkt als Binärzeichenfolge in ein Feld der Assertion einzubetten. Diese Datenboxen werden im Datenboxspeicher abgelegt, wobei jede einzelne Box ein einzelnes CBOR-Inhaltstyp-Feld (`cbor`) darstellt.

Die Daten einer Datenbox werden direkt als Wert des Datenfelds bereitgestellt, das ein `bstr` ist, sodass beliebige Binärdaten bereitgestellt werden können. Der Datentyp muss mithilfe des Felds „`dc:format`“ mit einem Standard-IANA-Medientyp identifiziert werden.

HINWEIS	IANA- strukturierte Suffixe (https://www.iana.org/assignments/media-type-structured-suffix/media-type-structured-suffix.xhtml) wie `+json` und `+zip` werden ebenfalls als Werte des Feldes `dc:format` unterstützt.

Manchmal kann es auch erforderlich sein, einen oder mehrere Asset-Typen als Wert für das Feld „`data_types`“ anzugeben, um das Format und die Verwendung dieser Daten klarer zu machen.

Ein Datenfeld muss die Bezeichnung „`c2pa.data`“ tragen und den Regeln für Assertion-Bezeichnungen in Bezug auf mehrere Instanzen entsprechen.

7.2. Schema und Beispiel

Das Schema für diesen Typ wird durch die Regel „`data-box-map`” in der CDDL-Definition in CDDL für Datenboxen definiert:

CDDL für Datenbox

```
; Feld zur Speicherung beliebiger Daten

data-box-map = {
  „dc:format”: format-string, ; IANA-Medientyp der Daten „data”: bstr, ;
  beliebige Text-/Binärdaten
  ? „data_types”: [1* $asset-type-map], ; zusätzliche Informationen zum Datentyp
}
```

Kapitel 8. Eindeutige Identifikatoren

8.1. Eindeutige Identifizierung von C2PA-Manifesten und Assets

Jedes C2PA-Manifest wird durch einen Uniform Resource Name RFC 8141, URNs aus dem c2pa-URN-Namensraum eindeutig identifiziert und referenziert, und ein C2PA-Asset wird durch den c2pa-URN-Wert seines aktiven Manifests eindeutig identifiziert. Die ABNF für die C2PA-URN wird durch die ABNF für C2PA-URN beschrieben.

Eine c2pa-URN muss aus zwei obligatorischen und zwei optionalen Komponenten in der folgenden Reihenfolge bestehen, wobei zwischen den einzelnen Abschnitten ein Doppelpunkt (`:`) steht.

- URN-Kennung (`urn:c2pa`): ERFORDERLICH.
- UUID v4 in Zeichenfolgenform (gemäß RFC 9562, Abschnitt 4): ERFORDERLICH.
- Claim Generator-Kennung: OPTIONAL.
- Zeichenfolge für Version und Grund (wie unten beschrieben): OPTIONAL.

Wenn vorhanden, darf die Zeichenfolge „Claim Generator identifier" aus maximal 32 Zeichen aus dem ASCII-Bereich (gemäß RFC 20) bestehen, die jedoch keine Steuerzeichen (RFC 20, 5.2) oder Grafikzeichen (RFC 20, 5.3) sein dürfen.

Wenn vorhanden, muss die Zeichenfolge „Version und Grund" aus einer positiven ganzen Zahl bestehen, gefolgt von einem Unterstrich (_) und einer weiteren positiven ganzen Zahl. Die Details zu diesen Werten und ihre Verwendung werden unter „Versionsmanifeste aufgrund von Konflikten" beschrieben. Wenn eine Zeichenfolge „Version und Grund" vorhanden ist, muss außerdem eine Zeichenfolge „Claim Generator Identifier" vorhanden sein, die jedoch leer sein kann.

ABNF für C2PA-URN

```
c2pa_urn = c2pa-namespace UUID [claim-generator [version-reason]] c2pa-

namespace = „urn:c2pa:"

;  diese  Definition  stammt  aus  RFC  9562  UUID
= 4hexOctet „-"
              2hexOctet    „-"
              2hexOctet    „-"
              2hexOctet    „-"
              6hexOctet
hexOctet = HEXDIG HEXDIG
DIGIT      = %x30-39
HEXDIG     = DIGIT / „A" / „B" / „C" / „D" / „E" / „F"

; ASCII, jedoch keine Steuerzeichen oder Grafikzeichen sichtbare-
Zeichen-außer-Leerzeichen = %x21-7E / %x80-FF

; claim-generator-identifier ist eine Zeichenfolge aus 0 bis 32 sichtbaren Zeichen außer Leerzeichen
; das bedeutet, dass eine leere Zeichenfolge gültig ist
claim-generator = ":" claim-generator-identifier
claim-generator-identifier = 0*32visible-char-except-space

; version-reason ist eine Zeichenfolge, die aus einer positiven ganzen Zahl besteht
; gefolgt von einem Unterstrich und einer positiven ganzen Zahl
```

```
version-reason = ":" version "_" reason version =
1*DIGIT
reason = 1*DIGIT
```

Beispiele:

- urn:c2pa:F9168C5E-CEB2-4FAA-B6BF-329BF39FA1E4
- urn:c2pa:F9168C5E-CEB2-4FAA-B6BF-329BF39FA1E4:acme
- urn:c2pa:F9168C5E-CEB2-4FAA-B6BF-329BF39FA1E4:acme:2_1
- urn:c2pa:F9168C5E-CEB2-4FAA-B6BF-329BF39FA1E4::2_1

HINWEIS Frühere Versionen dieser Spezifikation verwendeten RFC 9562, UUIDs URN und hatten die Kennung des Anspruchs Generator am Anfang der URN. Dies entsprach jedoch weder RFC 9562 (UUIDs) noch RFC 8141 (URNs).

Dieser c2pa-URN-Identifikator wird in verschiedenen Teilen eines C2PA-fähigen Workflows verwendet, beispielsweise bei der Identifizierung eines Assets als Bestandteil eines abgeleiteten oder zusammengesetzten Assets.

8.2. Versionsverwaltung aufgrund von Konflikten

Es können Situationen auftreten, in denen es aufgrund eines Konflikts zwischen Identifikatoren erforderlich ist, ein C2PA-Manifest neu zu kennzeichnen. Wenn beispielsweise ein Anspruchgenerator bereits ein Inhaltsstoffmanifest in den C2PA-Manifest-Speicher des Assets hinzugefügt hat und später einen weiteren Inhaltsstoff hinzufügt, dessen Manifest im Manifest-Speicher dieselbe Kennzeichnung aufweist, diese neuere Version des Manifests jedoch beispielsweise aufgrund einer Manipulation eines seiner Assertion-Werte unterschiedlich ist. In einem solchen Fall muss die geänderte Version des Inhaltsstoffmanifests in den C2PA-Manifest-Speicher des Assets kopiert und neu gekennzeichnet werden.

Um ein Manifest neu zu kennzeichnen:

- Wenn die aktuelle URN keine „Claim Generator Identifier String" enthält, muss der Claim Generator eine : an.
- In allen Fällen muss der Anspruchgenerator ein : an die URN anhängen, gefolgt von einer monoton steigenden Ganzzahl, beginnend mit 1, gefolgt von einem Unterstrich (_) und dann einer Ganzzahl aus der folgenden Liste, die den Grund für die Neukennzeichnung angibt.
  - 1: Konflikt mit einem anderen C2PA-Manifest

Wenn der Anspruchgenerator beispielsweise ein C2PA-Manifest aufgrund eines Konflikts zum zweiten Mal neu kennzeichnen muss, lautet die angehängte Zeichenfolge :2_1.

8.3. Identifizieren von Nicht-C2PA-Assets

Bei der Arbeit mit Assets, die kein C2PA-Manifest enthalten, aber eingebettete XMP-Daten mit Werten für xmpMM:DocumentID und/oder xmpMM:InstanceID gemäß der Definition in XMP-Spezifikation Teil 2, 2.2, enthalten, werden diese Werte als Identifikatoren für das Asset verwendet.

Bei der Arbeit mit Assets, die kein C2PA-Manifest und keine eingebetteten XMP-Daten enthalten, kann der Claim-Generator eine beliebige Methode seiner Wahl verwenden, um ihnen eine eindeutige Kennung zuzuweisen.

8.4. URI-Referenzen

8.4.1. Standard-URIs

Alle Verweise auf Informationen im Manifest, unabhängig davon, ob sie intern in der Ressource (d. h. eingebettet) oder extern zur Ressource (z. B. in der Cloud) gespeichert sind, müssen über JUMBF-URI-Referenzen gemäß ISO 19566-5:2023, C.2 referenziert werden. Diese URIs werden normalerweise entweder als Teil einer `hashed_uri`- oder einer hashed_ext_uri-Datenstruktur verwendet.

Wenn es sich um einen Verweis auf ein komprimiertes Manifest handelt, darf die JUMBF-URI keine Angaben zur Brob-Box enthalten, aber die URI zum Manifest wird so behandelt, als wäre das Manifest nicht komprimiert. Das bedeutet, dass die URI die Bezeichnung der `c2ma-` oder c2um-Box enthalten würde, nicht jedoch die Bezeichnung der c2cm-Box. Darüber hinaus darf die URI-Referenz auf ein komprimiertes Manifest nicht die Bezeichnung der Brob-Box enthalten, sondern nur die Bezeichnung des komprimierten Manifests selbst.

Bei der Auflösung einer internen JUMBF-URI-Referenz muss ein Validator die Referenz als ungelöst behandeln, wenn eine Bezeichnung im Pfad aufgrund mehrerer untergeordneter Felder mit derselben Bezeichnung mehrdeutig ist.

8.4.2. Hash-URIs

8.4.2.1. Eingebettet

Ein `hashed_uri` wird verwendet, wenn die URI für etwas steht, das in denselben C2PA Manifest Store eingebettet ist.

Diese Spezifikation bietet eine äquivalente Hash–URI-Map-Datenstruktur (in CDDL für Hash-URI) für Schemata, die eine CDDL-Definition verwenden:

CDDL für Hash-URI

```
; Die Datenstruktur, die zum Speichern einer Referenz auf eine URL innerhalb desselben JUMBF und deren Hash
verwendet wird. Wir verwenden hier einen Socket/Plug, damit hashed-uri-map in einzelnen Dateien verwendet
werden kann, ohne dass die Zuordnung in derselben Datei definiert sein muss
$hashed-uri-map /= {
  „url”: jumbf-uri-type, ; JUMBF-URI-Referenz
  ? „alg“: tstr .size (1..max-tstr-length), ; Eine Zeichenfolge, die den kryptografischen Hash-Algorithmus
identifiziert, der zur Berechnung aller Hashes in diesem Anspruch verwendet wird, entnommen aus der C2PA-Hash-
Algorithmus-Identifikatorliste. Wenn dieses Feld fehlt, wird der Hash-Algorithmus aus einer umgebenden Struktur
übernommen, wie sie durch diese Struktur definiert ist. Wenn beide vorhanden sind, wird das Feld in dieser
Struktur verwendet. Wenn an keiner dieser Stellen ein Wert vorhanden ist, ist diese Struktur ungültig; es gibt
keinen Standardwert.
  „hash”: bstr, ; Byte-Zeichenkette, die den Hashwert enthält
}

; mit CBOR Kopf (#) und Ende ($) werden in regulären Ausdrücken eingeführt, sodass sie nicht explizit
benötigt werden jumbf-uri-type /= tstr .regexp "self#jumbf=[\\w\\d\/][\\w\\d\\.\/:-]+[\\w\\d]"
```

Da Assertion Stores sich in derselben C2PA Manifest Box befinden müssen wie die Claims, auf die sie sich beziehen, sind nur `self#jumbf`-URIs zulässig. Diese `self#jumbf`-URIs können relativ zum gesamten C2PA-Manifest-Speicher sein, in

in diesem Fall müssen sie mit einem / (U+002F, Schrägstrich) beginnen, oder relativ zum aktuellen C2PA-Manifest. URIs dürfen nicht die Sequenz .. (ein Paar U+002E, Punkt) enthalten.

Beispiel 1. Beispiel für self#jumbf*-URIs*

Im Folgenden finden Sie Beispiele für gültige self#jumbf-URIs:

- self#jumbf=/c2pa/urn:c2pa:F095F30E-6CD5-4BF7-8C44-CE8420CA9FB7/c2pa.assertions/c2pa.thumbnail.claim bezieht sich auf den gesamten Speicher (da es mit / beginnt);
- self#jumbf=c2pa.assertions/c2pa.thumbnail.claim wäre relativ zum Manifest der Box, die die URI enthält.

8.4.2.2. Extern

Wenn auf eine Ressource verwiesen wird, die außerhalb des C2PA-Manifest-Speichers existiert, wird eine Hash-ext-uri-map-Datenstruktur verwendet. Es handelt sich um eine Variante der Hash-uri, da sie auf eine externe URI anstelle einer self#jumbf verweist. Die Hash-ext-uri-Datenstruktur wird durch die Hash-ext-uri-map-Regel in der folgenden CDDL in CDDL für Hash-externe URI definiert:

CDDL für gehasht-ext-uri

```
; Die Datenstruktur, die zum Speichern einer Referenz auf eine externe URL und deren Hash verwendet wird.
; Wir verwenden hier einen Socket/Plug, damit hashed-ext-uri-map in einzelnen Dateien verwendet werden kann
; ohne dass die Zuordnung in derselben Datei definiert ist
$hashed-ext-uri-map /= {
   "url": ext-url-type, ; http/https-URI-Referenz
   „alg”: tstr .size (1..max-tstr-length), ; Eine Zeichenfolge, die den kryptografischen Hash-Algorithmus
identifiziert, der zur Berechnung des Hashwerts der Daten dieser URI verwendet wird, entnommen aus der C2PA-
Hash-Algorithmus-Identifikatorliste. Im Gegensatz zu alg-Feldern in anderen Typen ist dieses Feld hier
obligatorisch.
   „hash”: bstr, ; Byte-Zeichenkette, die den Hashwert enthält
   ? „dc:format”: format-string, ; IANA-Medientyp der Daten
   ? „size“: size-type, ; Anzahl der Datenbytes
   ? „data_types”: [1* $asset-type-map], ; zusätzliche Informationen zum Datentyp
}

; mit CBOR Kopf (#) und Ende ($) werden in regulären Ausdrücken eingeführt, sodass sie nicht explizit benötigt
werden
ext-url-type /= tstr .regexp „https?:\/\/[-a-zA-Z0-9@:%._\\+~#=]{2,256}\\.[a-z]{2,6}\\b[-a-zA-Z0-
9@:%_\\+.~#?&//=]*“
```

WICHTIG

Entsprechend der gängigen Praxis wird empfohlen, das https-Schema zum Abrufen von Assertion-Daten zu verwenden, um die Privatsphäre der übertragenen Daten zu schützen. http ist jedoch ebenfalls zulässig, da die Integrität der Daten durch das Hash-Feld geschützt ist und dieser Datenschutz möglicherweise nicht in allen Fällen erforderlich ist. Autoren von Manifesten mit externen URIs sollten das Schema entsprechend ihren Anforderungen auswählen.

Das optionale Feld „dc:format“ bietet, sofern vorhanden, eine Alternative zum Feld „Content-Type“ der HTTP(S)-Header. Ist dieses Feld vorhanden, muss es als erforderliches Format verwendet werden, das bei jeder Inhaltsaushandlung/Anfrage abgerufen wird.

Manchmal kann es auch erforderlich sein, einen oder mehrere Asset-Typen als Wert des Feldes „`data_types`“ anzugeben, um mehr Klarheit über das Format und die Verwendung dieser Daten zu schaffen.

Ein optionales Größenfeld ist ebenfalls vorhanden, um die Größe der abzurufenden Daten anzugeben. Dies kann für einen Validierer zusätzlich zum Hash als Hinweis nützlich sein.

HINWEIS Es könnte verwendet werden, um Informationen darüber bereitzustellen, ob ein Download oder eine Validierung oder beides durchgeführt werden soll.

8.4.2.3. Hashing von JUMBF-Boxen

Bei der Erstellung einer URI-Referenz zu einer beliebigen JUMBF-Box (z. B. Assertions- und Datenboxen) muss der Hash über den Inhalt der JUMBF-Superbox der Struktur durchgeführt werden, die sowohl die JUMBF-Beschreibungsbox als auch alle darin enthaltenen Inhaltsboxen umfasst (jedoch nicht den Header der JUMBF-Superbox der Struktur).

HINWEIS Weitere Einzelheiten zum Hashing finden Sie in Abschnitt 13.1, „Hashing“.

Wie in der neuesten Version von JUMBF (ISO 19566-5:2023) beschrieben und in Abbildung 4 „Beispiel für `eine c2pa.`actions-Assertion“ dargestellt, kann ein neues `privates` Feld als Teil eines beliebigen JUMBF-Beschreibungsfeldes vorhanden sein. Diese C2PA-Spezifikation definiert das C2PA-Salt als `privates` Feld, dessen Wert ein Standardfeld ist, das aus folgenden Elementen besteht:

- einer Feldlänge (LBox, als 4-Byte-Big-Endian-Ganzzahl ohne Vorzeichen);
- einem Feldtyp (TBox, 4-Byte-Big-Endian-Ganzzahl ohne Vorzeichen, mit dem Wert `c2sh` (für C2PA-Salt-Hash));
- und Nutzdaten (bestehend aus zufällig generierten Binärdaten mit einer Länge von entweder 16 oder 32 Byte).

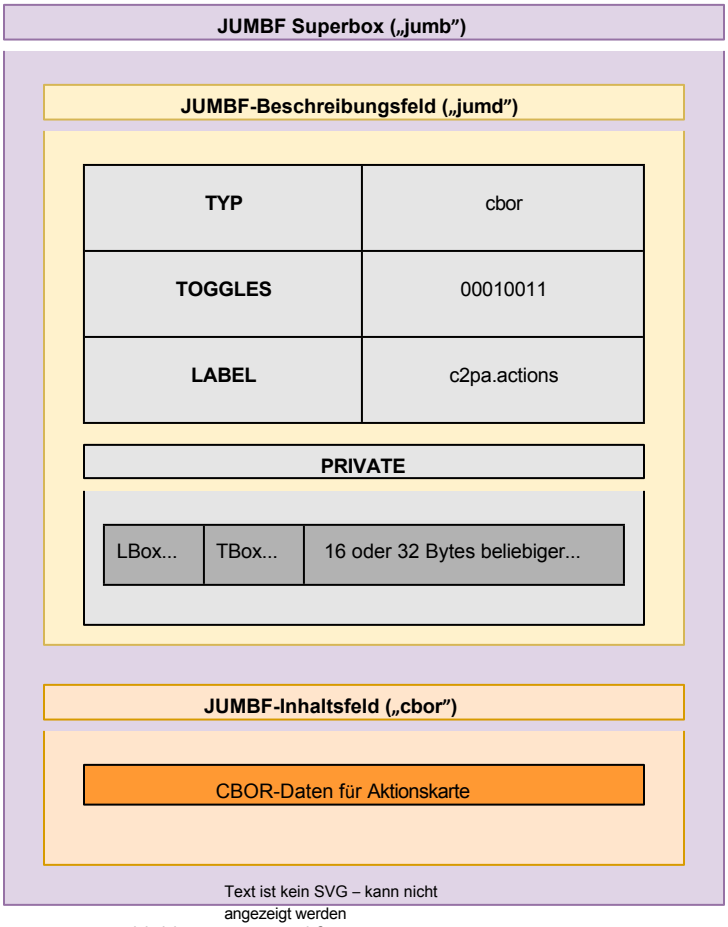

Abbildung 4. Beispiel für eine `c2pa.actions`-Assertion

Kapitel 9. Bindung an Inhalte

9.1. Übersicht

Ein wichtiger Aspekt des Standard-C2PA-Manifests ist das Vorhandensein einer oder mehrerer Datenstrukturen, sogenannter Inhaltsbindungen, die Teile des Assets eindeutig identifizieren können. Es gibt zwei Arten von Bindungen, die von C2PA unterstützt werden: harte Bindungen und weiche Bindungen. Eine harte Bindung (auch als kryptografische Bindung bezeichnet) ermöglicht es dem Validierer, sicherzustellen, dass (a) dieses Manifest zu diesem Asset gehört und (b) das Asset nicht verändert wurde, indem Werte ermittelt werden, die nur zu diesem Asset und zu keinem anderen passen, auch nicht zu anderen davon abgeleiteten Assets oder daraus erstellten Wiedergaben. Eine weiche Bindung wird aus dem digitalen Inhalt eines Assets berechnet und nicht aus seinen Rohbits. Eine weiche Bindung ist nützlich, um abgeleitete Assets und Asset-Wiedergaben zu identifizieren.

Ein einzelnes Manifest darf nicht mehr als eine Aussage enthalten, die eine feste Bindung definiert, kann jedoch null oder mehrere Aussagen enthalten, die weiche Bindungen definieren.

9.2. Feste Bindungen

9.2.1. Hashing unter Verwendung von Byte-Bereichen

Die einfachste Art der harten Bindung, die zur Erkennung von Manipulationen verwendet werden kann, ist ein kryptografischer Hash-Algorithmus, wie in Abschnitt 13.1, „Hashing“, beschrieben, der auf einige oder alle Bytes eines Assets angewendet wird. Dieser Ansatz kann für jede Art von Asset verwendet werden, sollte jedoch nur für Formate in Betracht gezogen werden, die keine der Formen des boxbasierten Hashings unterstützen.

Bei Verwendung dieser Form der harten Bindung wird eine Daten-Hash-Assertion verwendet, um den Bereich der Bytes zu definieren, die gehasht werden (und diejenigen, die nicht gehasht werden). Da eine Daten-Hash-Assertion einen Byte-Bereich definiert, ist sie flexibel genug, um verwendet werden zu können, unabhängig davon, ob es sich bei dem Asset um eine einzelne Binärdatei handelt oder ob es in mehreren Blöcken oder Teilen dargestellt wird.

9.2.2. Hashing mit einem allgemeinen Box-Hash

Wenn das Format einer Ressource ein nicht auf BMFF basierendes Box-Format ist, wie beispielsweise JPEG, PNG, GIF oder andere hier aufgeführte Formate, sollte eine allgemeine Box-Hash-Assertion verwendet werden. Diese Assertion besteht aus einem Array von Strukturen, von denen jede eine oder mehrere Boxen (nach ihrem Namen/ihrer Kennung) und einen Hash auflistet, der die Daten dieser Boxen (und alle möglichen Daten, die in der Datei zwischen ihnen vorhanden sein können) abdeckt, zusammen mit dem für das Hashing verwendeten Algorithmus.

9.2.3. Hashing einer BMFF-formatierten Ressource

Wenn das Asset auf ISO BMFF basiert, kann stattdessen eine für das boxbasierte Format optimierte Hard-Binding-Assertion (sogenannte BMFF-basierte Hash-Assertions) verwendet werden.

Bei einer monolithischen MP4-Datei, bei der die mdat-Box als Einheit validiert wird, erfolgt die Validierung der Assertion fast identisch mit einer Daten-Hash-Assertion. Es wird lediglich eine Box-Ausschlussliste anstelle von Byte-Bereichen verwendet, um den Bereich der Bytes zu definieren, die gehasht werden (und diejenigen, die nicht gehasht werden).

Bei fragmentierten MP4-Dateien (fMP4) muss die Assertion selbst mit chunk-spezifischen Hash-Informationen kombiniert werden, die sich gemäß Abschnitt A.5, „Einbetten von Manifesten in BMFF-basierte Assets“, befinden.

9.2.4. Hashing einer Sammlung

In Workflows, in denen sich das C2PA-Manifest auf eine Sammlung von Assets statt auf ein einzelnes Asset bezieht, muss die Hash-Assertion der Sammlung als Methode zur Angabe der festen Bindungen für die Assets in der Sammlung verwendet werden.

HINWEIS	Eine Hash-Assertion für Sammlungsdaten kann beispielsweise verwendet werden, um jeden Ordner eines Trainingsdatensatzes für ein KI-/ML-Modell zu beschreiben.

9.2.5. Asset-Metadaten-Bindungen

Der Anspruchgenerator kann Metadaten von Assets (d. h. Metadaten außerhalb eines C2PA-Manifests wie EXIF oder XMP) aus der Inhaltsbindung ausschließen. Dazu muss er die entsprechenden Ausschlussmechanismen für Daten-Hash-Assertions, allgemeine Box-Hash-Assertions oder BMFF-basierte Hash-Assertions verwenden.

HINWEIS	Ausgeschlossene Metadaten von Assets werden dem Unterzeichner nicht zugeordnet.

Alle Asset-Metadatenwerte, die von der gemeinsamen Metadaten-Assertion unterstützt werden, wie in Anhang B, *Implementierungsdetails für* `c2pa.metadata`, beschrieben, und die vom Unterzeichner bestätigt werden können, sollten in eine solche Assertion kopiert und in das C2PA-Manifest aufgenommen werden.

9.3. Weiche Bindungen

9.3.1. Allgemeines

Weiche Bindungen werden mithilfe von weichen Bindungsaussagen beschrieben, wie beispielsweise einem aus dem digitalen Inhalt berechneten Fingerabdruck oder einem unsichtbaren Wasserzeichen, das in den digitalen Inhalt eingebettet ist. Diese weichen Bindungen ermöglichen es, digitale Inhalte auch dann abzugleichen, wenn sich die zugrunde liegenden Bits unterscheiden.

HINWEIS	Beispielsweise eine Asset-Wiedergabe in einer anderen Auflösung oder einem anderen Codierungsformat.

Wenn außerdem ein C2PA-Manifest aus einem Asset entfernt wird, aber eine Kopie dieses Manifests in einem Provenienzspeicher an anderer Stelle verbleibt, können das Manifest und das Asset mithilfe der verfügbaren Soft Bindings abgeglichen werden.

Da sie einem anderen Zweck dienen, darf eine weiche Bindung nicht als harte Bindung verwendet werden.

9.3.2. Liste der zulässigen Soft-Binding-Algorithmen

Alle Soft-Bindungen müssen unter Verwendung eines der in der Liste der Soft-Bindungsalgorithmen aufgeführten Algorithmen generiert werden, die von dieser Spezifikation unterstützt werden.

Kapitel 10. Ansprüche

10.1. Übersicht

Ein **Anspruch** fasst alle Aussagen zu einem Vermögenswert zu einem bestimmten Zeitpunkt zusammen, einschließlich der Aussagen zur Bindung an den Inhalt. Der Anspruch wird dann kryptografisch gehasht und signiert, wie in Abschnitt 10.3.2.4, „Signieren eines Anspruchs“, beschrieben. Ein Anspruch hat dieselben Eigenschaften wie eine Behauptung, einschließlich der Zuweisung der Kennzeichnung (`c2pa.claim.v2`), unterstützt jedoch nicht die Verwendung von Behauptungsmetadaten. Ein Anspruch wird als CBOR-Daten codiert und muss daher den grundlegenden deterministischen Codierungsanforderungen von CBOR entsprechen (siehe RFC 8949, Abschnitt 4.2.1).

HINWEIS Frühere Versionen unterstützten die Verwendung von Assertion-Metadaten mit Ansprüchen, dies ist jedoch inzwischen veraltet.

Eine frühere Version dieser Spezifikation verwendete das Label `c2pa.claim` und `die` zugehörige `Claim-Map` für den Claim, aber diese sind nun veraltet. Validatoren sollten dieses Label (und `die` zugehörige `Claim-Map`) weiterhin akzeptieren, aber Claim-Generatoren dürfen einen solchen Claim nicht mehr erstellen.

10.2. Syntax

10.2.1. Schema

Das Schema für diesen Typ wird durch die Regeln `claim-map-v2` und `claim-map` in der folgenden CDDL-Definition für Ansprüche mit den Bezeichnungen `c2pa.claim.v2` bzw. `c2pa.claim` definiert:

```
; CDDL-Schema für eine Anspruchskarte in
C2PA claim-map = {
  „claim_generator”: tstr, ; Eine User-Agent-Zeichenfolge, formatiert gemäß
http://tools.ietf.org/html/rfc7231#section-5.5.3, zur Angabe des Namens und der Version des
Anspruchsgenerators, der den Anspruch erstellt hat
  „claim_generator_info”: [1* generator-info-map],
  „signature”: jumbf-uri-type, ; JUMBF-URI-Verweis auf die Signatur dieses Anspruchs „assertions”: [1* $hashed-
  uri-map],
  „dc:format”: tstr, ; Medientyp des Assets
  „instanceID”: tstr .size (1..max-tstr-length), ; identifiziert eindeutig eine bestimmte Version eines Assets
  ? „dc:title”: tstr .size (1..max-tstr-length), ; Name des Assets,
  ? „redacted_assertions”: [1* jumbf-uri-type], ; Liste der JUMBF-URI-Referenzen auf die Assertions der
redigierten Inhaltsverzeichnisse
  ? „alg“: tstr .size (1..max-tstr-length), ; Eine Zeichenfolge, die den kryptografischen Hash-Algorithmus
identifiziert, der zur Berechnung aller in diesem Anspruch aufgeführten Daten-Hash-Assertions verwendet wird,
sofern nicht anders angegeben, entnommen aus dem C2PA-Daten-Hash-Algorithmus-Identifikationsregister. Dies
liefert den Wert für das Feld „alg“ in den in diesem Anspruch enthaltenen Strukturen „data-hash“ und „hashed-uri“.
  ? „alg_soft”: tstr .size (1..max-tstr-length), ; Eine Zeichenfolge, die den Algorithmus identifiziert, der
zur Berechnung aller in diesem Anspruch aufgeführten Soft-Binding-Assertions verwendet wird, sofern nicht
anders angegeben, entnommen aus dem C2PA-Soft-Binding-Algorithmus-Identifikationsregister.
  ? „metadata”: $assertion-metadata-map, ; zusätzliche Informationen zur Assertion
}

; CDDL-Schema für eine Anspruchskarte in
C2PA claim-map-v2 = {
```

```
  „instanceID”: tstr .size (1..max-tstr-length), ; identifiziert eindeutig eine bestimmte Version eines Assets
  „claim_generator_info”: $generator-info-map, ; der Anspruchgenerator dieses Anspruchs „signature”: jumbf-uri-
  type, ; JUMBF-URI-Verweis auf die Signatur dieses Anspruchs „created_assertions”: [1* $hashed-uri-map],
  ? „gathered_assertions”: [1* $hashed-uri-map],
  ? „dc:title”: tstr .size (1..max-tstr-length), ; Name des Assets,
  ? „redacted_assertions”: [1* jumbf-uri-type], ; Liste der JUMBF-URI-Referenzen zu den redigierten Aussagen der
Inhaltsverzeichnisse
  ? „alg”: tstr .size (1..max-tstr-length), ; Eine Zeichenfolge, die den kryptografischen Hash-Algorithmus
identifiziert, der zur Berechnung aller in diesem Anspruch aufgeführten Daten-Hash-Assertions verwendet wird,
sofern nicht anders angegeben, entnommen aus dem C2PA-Daten-Hash-Algorithmus-Identifikationsregister. Dies
liefert den Wert für das Feld „alg” in den in diesem Anspruch enthaltenen Strukturen „data-hash” und „hashed-uri”.
  ? „alg_soft”: tstr .size (1..max-tstr-length), ; Eine Zeichenfolge, die den Algorithmus identifiziert, der
zur Berechnung aller in diesem Anspruch aufgeführten Soft-Binding-Assertions verwendet wird, sofern nicht
anders angegeben, entnommen aus dem C2PA-Soft-Binding-Algorithmus-Identifikationsregister.
  ? „metadata”: $assertion-metadata-map, ; (VERALTET) zusätzliche Informationen zur Assertion
}

generator-info-map = {
  „name“: tstr .size (1..max-tstr-length), ; Eine für Menschen lesbare Zeichenfolge, die den Anspruchgenerator
benennt
  ? „version”: tstr, ; Eine für Menschen lesbare Zeichenfolge der Produktversion
  ? „icon”: $hashed-uri-map / $hashed-ext-uri-map, ; Hash-URI zum Symbol (entweder eingebettet oder remote)
  ? „operating_system”: tstr, ; Eine für Menschen lesbare Zeichenfolge des Betriebssystems, auf dem der
Anspruchgenerator ausgeführt wird
  * tstr => beliebig
}
```

Ein Beispiel für die Struktur `von claim-map-v2` in CBOR-Diagnoseschrift (RFC 8949, Abschnitt 8):

```
{
  „alg“: „sha256“,
  „claim_generator_info“: {
    „name“: „Joe's Photo Editor“,
    „version“: „2.0“,
    „Betriebssystem“: „Windows 10“
  },
  „signature“: „self#jumbf=c2pa.signature“,
  „created_assertions“: [
    {
      „URL“: „self#jumbf=c2pa.assertions/c2pa.hash.data“, „Hash“:
      b64'U9Gyz05tmpftkoEYP6XYNsMnUbnS/KcktAg2vv7n1n8='
    },
    {
      „url“: „self#jumbf=c2pa.assertions/c2pa.thumbnail.claim“, „hash“:
      b64'G5hfJwYeWTlflxOhmfCO9xDAK52aKQ+YbKNhRZeq92c='
    },
    {
      "url": "self#jumbf=c2pa.assertions/c2pa.ingredient.v3", "hash":
      b64'Yzag4o5jO4xPyfANVtw7ETlbFSWZNfeM78qbSi8Abkk='
    }
  ],
  „redacted_assertions” : [ „self#jumbf=/c2pa/urn:c2pa:5E7B01FC-
    4932-4BAB-AB32-
D4F12A8AA322/c2pa.assertions/c2pa.metadata"
  ]
}
```

10.2.2. Felder

Wenn vorhanden, muss der Wert von `dc:title` ein für Menschen lesbarer Name für das Asset sein.

HINWEIS Das Feld „`dc:format`“ ist in `c2pa.claim` nicht mehr vorhanden.

Wenn das Asset XMP enthält, sollte die `xmpMM:InstanceID` des Assets als `instanceID` verwendet werden. Wenn kein XMP verfügbar ist, muss eine andere eindeutige Kennung für das Asset als Wert für `instanceID` verwendet werden.

HINWEIS Einige Feldnamen, wie z. B. `dc:title`, haben Namespace-Präfixe, da ihre Namen und Definitionen direkt aus dem XMP-Standard übernommen wurden. Ihre Verwendung in C2PA erfordert jedoch nicht die Verwendung von XMP.

Das Signaturfeld muss vorhanden sein und eine URI-Referenz zu einer Anspruchsignatur enthalten.

Das Feld „`created_assertions`“ muss vorhanden sein und eine oder mehrere URI-Referenzen zu den durch diesen Anspruch gemachten Aussagen enthalten. In einem Standardmanifest muss es mindestens eine Referenz zu einer Aussage enthalten, die eine feste Bindung darstellt, sowie eine Referenz zu einer Handlungsaussage.

HINWEIS Alle `created_assertions` werden dem Unterzeichner zugeschrieben, da das Vertrauensmodell auf dem Vertrauen in den Unterzeichner basiert.

Wenn vorhanden, muss das Feld „`gathered_assertions`“ eine oder mehrere URI-Referenzen zu Assertions enthalten, die dem Anspruchgenerator von anderen Komponenten im Workflow bereitgestellt wurden.

HINWEIS Durch die Aufnahme einer Assertion in diese Liste erklärt der Anspruchgenerator, dass die Assertion Teil der , aber nicht vom Anspruchgenerator stammt und nicht dem Unterzeichner zugeschrieben wird. Beispielsweise würden Assertions, die von einem menschlichen Akteur eingegebene Informationen enthalten, in `gathered_assertions` aufgeführt werden.

Wenn vorhanden, muss das Feld „`redacted_assertions`“ eine oder mehrere URI-Referenzen zu redigierten Assertions enthalten.

10.2.3. Informationen zum Anspruchgenerator

10.2.3.1. Allgemeines

Detaillierte Informationen zum Anspruchgenerator müssen als Wert von `claim_generator_info` vorhanden sein. Ein Manifest-Verbraucher muss den Wert von `claim_generator_info` verwenden, um Informationen über den Anspruchgenerator für sich selbst oder zur Darstellung in einer Benutzeroberfläche zu ermitteln.

HINWEIS `c2pa.claim` verfügt über ein Feld `claim_generator`, dessen Wert eine einfache Zeichenfolge ist, die in `c2pa.claim.v2` nicht mehr vorhanden ist.

10.2.3.2. Generator-Info-Map

Beim Hinzufügen eines Feldes `claim_generator_info` ist dessen Wert ein `Generator-Info`-Map-Objekt, das ein Feld name-Feld enthalten. Es kann auch entweder ein version-Feld oder ein icon-Feld oder beides enthalten. Darüber hinaus sind alle anderen Felder

unter Verwendung des in Abschnitt 6.2.1, „Namespacing“, beschriebenen standardmäßigen entitätsspezifischen Namespacing zulässig. Die Daten in diesem Objekt müssen den nicht-menschlichen (Hardware- oder Software-)Akteur repräsentieren, der den Anspruch tatsächlich generiert hat (auch bekannt als der Anspruchsgenerator selbst).

Ein Anspruchgenerator kann eine grafische Darstellung seiner selbst, hier als `Symbol` bezeichnet, für einen Manifest-Verbraucher bereitstellen, der eine Benutzererfahrung präsentiert. Der Wert des Symbolfelds, sofern vorhanden, muss eine gehashte URI sein. Diese gehashte URI muss zu einer eingebetteten Datenaussage mit der Bezeichnung `c2pa.icon` führen und den Regeln für Aussagekennzeichnungen in Bezug auf mehrere Instanzen entsprechen. Manifest-Konsumenten sollten auch den in früheren Versionen dieser Spezifikation empfohlenen Datenbox-Ansatz unterstützen.

HINWEIS Wie beim Assertions-Array wird der für eine Hash-URI verwendete Hash-Algorithmus durch das Feld „alg” in der Hash-URI bestimmt oder, falls dieses fehlt, durch ein Feld „alg” im Claim.

Beispiel unter Verwendung von Anspruchgenerator-Informationen

```
{
     „claim_generator_info” : { „name”: „Joe's
          Photo Editor”, „version”: „2.0”,
          "operating_system": "Windows 10", "icon": {
               "url": "http://cdn.examplephotoagency.com/logo.svg", "hash":
               "5bdec8169b4e4484b79aba44cee5c6bd"
          }
     }
}
```

10.3. Erstellen einer Behauptung

10.3.1. Erstellen von Assertions

Bevor der Anspruch finalisiert werden kann, müssen alle Assertions erstellt und in einem neu erstellten C2PA Assertion Store gespeichert werden, wie später in diesem Dokument beschrieben.

Beim Erstellen eines Standardmanifests ist es unter Umständen nicht möglich, alle erforderlichen Bindungsinformationen zum Zeitpunkt der Anspruchserstellung zu kennen. In diesem Fall verwenden Sie die mehrstufige Verarbeitungsmethode, um die Informationen einzurichten und später auszufüllen.

10.3.2. Vorbereitung des Anspruchs

10.3.2.1. Hinzufügen von Assertions und Redaktionen

Der Anspruch muss ein Feld „created_assertions” enthalten und kann ein Feld „gathered_assertions” enthalten. Die kombinierten Werte dieser beiden Felder stellen eine Liste aller URI-Referenzen für alle Assertions dar, die zum Assertion Store hinzugefügt wurden und von diesem Anspruch „beansprucht” werden. In einem Standard-Manifest muss der Wert des Feldes „created_assertions” mindestens eine Assertion enthalten, die eine feste Bindung darstellt.

Wenn Aussagen in Inhaltsstoffangaben redigiert werden, müssen ihre URI-Referenzen zu der Liste hinzugefügt werden, die den Wert des Feldes „redacted_assertions" darstellt.

10.3.2.2. Hinzufügen von Inhaltsstoffen

In vielen Autorenszenarien erstellt ein Akteur kein völlig neues Asset, sondern bezieht andere vorhandene Assets ein, auf denen er seine Arbeit aufbaut – entweder als abgeleitetes Asset, zusammengesetztes Asset oder Asset-Wiedergabe. Diese vorhandenen Assets werden als „Ingredienzien" bezeichnet, und ihre Verwendung wird in den Herkunftsdaten durch eine Ingredienzien-Angabe dokumentiert.

Wenn eine Komponente ein oder mehrere C2PA-Manifeste enthält, müssen diese Manifeste in den C2PA-Manifest-Speicher dieser Komponente eingefügt werden, um sicherzustellen, dass die Herkunftsdaten intakt bleiben. Solche Komponentenmanifeste werden gemäß Abschnitt 11.1.4, „C2PA-Box-Details", zum JUMBF hinzugefügt. Wenn ein Manifest mit derselben eindeutigen Kennung bereits im C2PA-Manifest-Speicher vorhanden ist, werden die beiden verglichen (mittels Hashing). Wenn sie identisch sind, wird das neue Manifest ignoriert. Wenn sie unterschiedlich sind, wird das neue Manifest zum Speicher hinzugefügt, nachdem seine eindeutige Kennung gemäß Kapitel 8, *„Eindeutige Kennungen",* in einen neuen Wert geändert wurde.

Wenn das Manifest einer Zutat entfernt ist und der Anspruchgenerator das Manifest nicht abrufen kann, sollte er den Fehlercode „manifest.inaccessible" verwenden, um dies widerzuspiegeln.

10.3.2.3. Verbinden der Signatur

Die Signatur kann nicht Teil der signierten Nutzlast sein, aber da ihr Label vordefiniert ist, ist auch die vollständige URI-Referenz bekannt. Daher können wir diese in den Anspruch aufnehmen, indem wir den Wert des Signaturfelds des Anspruchs auf diese URI-Referenz setzen.

HINWEIS Dadurch wird die explizite Bindung des Anspruchs an seine Signatur hergestellt.

10.3.2.4. Signieren eines Anspruchs

Die Erstellung der Signatur ist in Abschnitt 13.2, „Digitale Signaturen", festgelegt.

Für beide Arten von Manifesten, Standard und Aktualisierung, muss das Payload-Feld von `Sig_structure` das serialisierte CBOR des Anspruchsdokuments sein und den Modus „Detached Content" verwenden.

Die aus dem digitalen Signaturverfahren resultierende serialisierte COSE_Sign1_Tagged-Struktur wird in das Feld „C2PA Claim Signature" geschrieben.

10.3.2.5. Zeitstempel

10.3.2.5.1. Verwendung von RFC 3161

Wenn möglich, sollte der Anspruchgenerator eine RFC 3161-konforme Zeitstempelbehörde (TSA) (RFC 3161) verwenden, um einen vertrauenswürdigen Zeitstempel zu erhalten, der belegt, dass die Signatur selbst tatsächlich zu einem bestimmten Datum und einer bestimmten Uhrzeit existierte, und diesen als Gegenunterschrift in die COSE_Sign1_Tagged-Struktur aufnehmen.

Anspruchsgeneratoren werden dazu angehalten, Zeitstempel zu erhalten und einzufügen, um sicherzustellen, dass ihre Manifeste gültig bleiben. Als

Wie in Kapitel 15, *Validierung*, beschrieben, verlieren Manifeste ohne Zeitstempel ihre Gültigkeit, wenn die Signaturberechtigung abläuft oder widerrufen wird. Ein Manifest darf nur einen Zeitstempel enthalten.

HINWEIS Frühere Versionen dieser Spezifikation erlaubten die Aufnahme mehrerer Zeitstempel in ein Manifest.

10.3.2.5.2. Auswahl der Nutzlast

In einer früheren Version dieser Spezifikation wurde für das Payload-Feld im Zeitstempel derselbe Wert verwendet wie in der `Sig_signature`, wie in Abschnitt 10.3.2.4, „Signieren eines Anspruchs“, beschrieben. Diese Nutzlast wird fortan als „v1-Nutzlast“ in einem „v1-Zeitstempel“ bezeichnet und gilt als veraltet. Ein Anspruchgenerator darf keine solche Nutzlast erstellen, aber ein Validator muss eine solche Nutzlast verarbeiten, wenn sie vorhanden ist.

Die „v2-Nutzlast“ des „v2-Zeitstempels“ ist der Wert des Signaturfelds der COSE_Sign1_Tagged-Struktur, die als Teil von Abschnitt 10.3.2.4, „Signieren eines Anspruchs“, erstellt wurde. Eine „v2-Nutzlast“ muss von Anspruchgeneratoren verwendet werden, die eine Zeitstempeloperation durchführen.

HINWEIS Der Wert des Signaturfelds umfasst den gesamten serialisierten `bstr`, einschließlich der Bytes, die den Haupttyp und die Länge angeben (nicht nur die Zeichenfolge selbst).

10.3.2.5.3. Erhalten des Zeitstempels

Alle Zeitstempel müssen gemäß RFC 3161 mit den folgenden zusätzlichen Anforderungen erstellt werden:

- Der `MessageImprint` der TimeStampReq-Struktur (RFC 3161, Abschnitt 2.4.1) muss durch Erstellen des ToBeSigned-Werts in RFC 8152, Abschnitt 4.4, mit den folgenden Werten für Elemente der `Sig_Struktur` berechnet werden:
  - Das Kontextelement muss `CounterSignature` sein.
  - Das Payload-Element muss der in Abschnitt 10.3.2.5.2, „Auswahl der Nutzlast“, beschriebene Wert sein.
  - Die übrigen Elemente von `Sig_structure` entsprechen der Beschreibung in Abschnitt 13.2.3, „Berechnung der Signatur”.
- Der Wert `„ToBeSigned“` wird dann mit einem Hash-Algorithmus aus der Liste der zulässigen Algorithmen in Abschnitt 13.1, „Hashing“, die von der TSA unterstützt werden, gehasht, und dieser Hash-Algorithmus und dieser Wert werden in den `MessageImprint` eingefügt. Wenn die TSA keinen der Hash-Algorithmen aus der Liste der zulässigen Algorithmen unterstützt, kann sie nicht für die Zeitstempelung verwendet werden.
  - Wenn möglich, sollte der Hash-Algorithmus denselben Hash-Algorithmus verwenden, der auch für die digitale Signatur des Anspruchs verwendet wird.
- Der boolesche Wert `„certReq“` der Struktur `„TimeStampReq“` muss in der Anfrage an die TSA angegeben werden, um sicherzustellen, dass die Zertifikatskette in der Antwort bereitgestellt wird.

10.3.2.5.4. Speichern des Zeitstempels

v1-Zeitstempel (veraltet) werden in einem ungeschützten COSE-Header gespeichert, dessen Bezeichnung die Zeichenfolge `sigTst` lautet. Falls vorhanden, muss der Wert dieses Headers ein `tstContainer` sein, der durch Beispiel 2, „CDDL für `tstContainer`“, definiert ist. Der Inhalt der TimeStampResp-Struktur, die als Antwort von der TSA empfangen wird, muss als Wert der val-Eigenschaft eines Elements von `tstTokens` gespeichert werden.

v2-Zeitstempel müssen in einem ungeschützten COSE-Header gespeichert werden, dessen Bezeichnung die Zeichenfolge `sigTst2` lautet. Wenn vorhanden, muss der Wert dieses Headers ein `tstContainer` sein, `der` durch Beispiel 2, „CDDL für `tstContainer`", definiert ist. Der Wert des Feldes `timeStampToken` der Struktur `TimeStampResp`, die als Antwort von der TSA empfangen wurde, muss als Wert der Eigenschaft `val` eines Elements von `tstTokens` gespeichert werden. Er muss als DER-codiertes RFC 3161 `TimeStampToken` formatiert sein, das in eine CBOR-Byte-Zeichenkette eingeschlossen ist.

HINWEIS

Ein v2-Zeitstempel entspricht dem „CTT"-Modell des COSE-Header-Parameters für RFC 3161 Time-Stamp Tokens Draft. Es erfordert, dass die vollständige Signaturstruktur vor der Zeitstempelung fertiggestellt ist, sodass der Zeitstempel als Gegenzeichnung für die gesamte Signaturstruktur, einschließlich des eigentlichen Zertifikats, dienen kann.

Wenn keine Zeitstempel enthalten sind, darf keiner der beiden Header (`sigTst` oder `sigTst2`) im ungeschützten COSE-Header vorhanden sein.

Beispiel 2. CDDL für `tstContainer`

```
; CBOR-Version von tstContainer und zugehörigen Strukturen basierend auf dem JSON-Schema unter
; https://forge.etsi.org/rep/esi/x19_182_JAdES/raw/v1.1.1/19182-jsonSchema.json tstContainer = {
  „tstTokens": [1* tstToken]
}

tstToken = { „val":
  bstr
}
```

HINWEIS

Die obige Definition ist eine CBOR-Anpassung einer Teilmenge des Schemas aus JAdES, Abschnitt 5.3.4, und seines JSON-Schemas, mit der Ausnahme, dass der Inhalt von „`val`" eine Byte-Zeichenkette und keine Base64-kodierte Zeichenkette ist.

10.3.2.6. Informationen zur Sperrung von Berechtigungsnachweisen

Wenn die Berechtigungsnachweise des Unterzeichners die Abfrage ihres Online-Status unterstützen und die Berechtigungsnachweise einen Verweis auf einen Dienst enthalten, der zeitgestempelte Informationen zum Status der Berechtigungsnachweise bereitstellt, sollte der Anspruchgenerator den Dienst abfragen, die Antwort erfassen und sie auf die im Vertrauensmodell für Berechtigungsnachweise beschriebene Weise speichern. Wenn Informationen zur Sperrung von Berechtigungsnachweisen auf diese Weise angehängt werden, muss nach der Signierung auch ein vertrauenswürdiger Zeitstempel eingeholt werden, wie in Abschnitt 10.3.2.5, „Zeitstempel", beschrieben.

10.3.3. Beispiele für Ansprüche

10.3.3.1. Einzelner Anspruch

Hier sehen Sie eine visuelle Darstellung eines Bildes, das eine einzelne Behauptung mit mehreren darin eingebetteten Aussagen enthält.

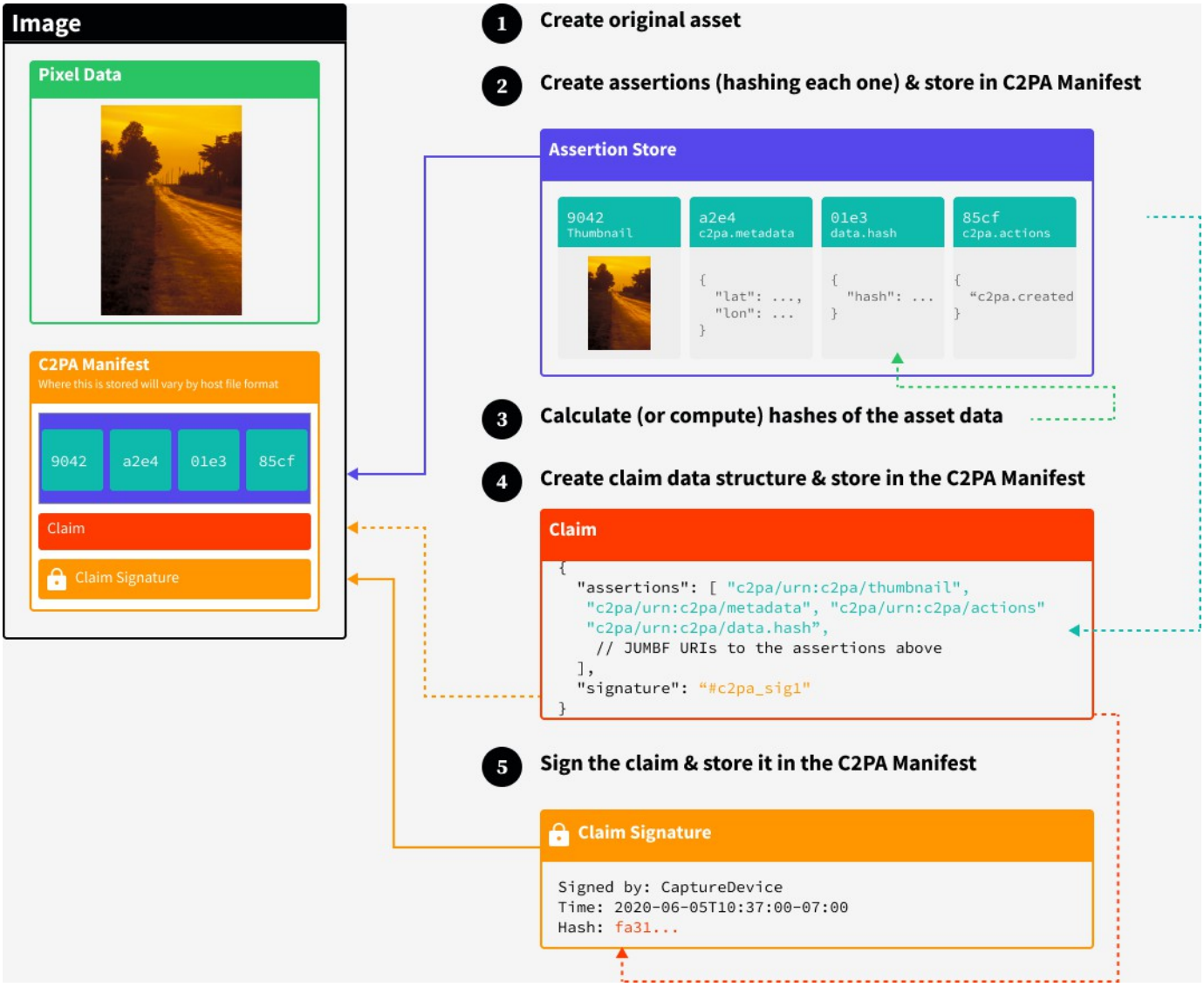

Abbildung 5. Eine einzelne Behauptung mit Aussagen

10.3.3.2. Mehrere Ansprüche

In diesem Beispiel für die Erstellung einer zweiten Behauptung zum vorherigen Beispiel wurde eine der ursprünglichen Aussagen aus der vorherigen Behauptung entfernt. Die visuelle Darstellung für dieses Szenario würde wie folgt aussehen:

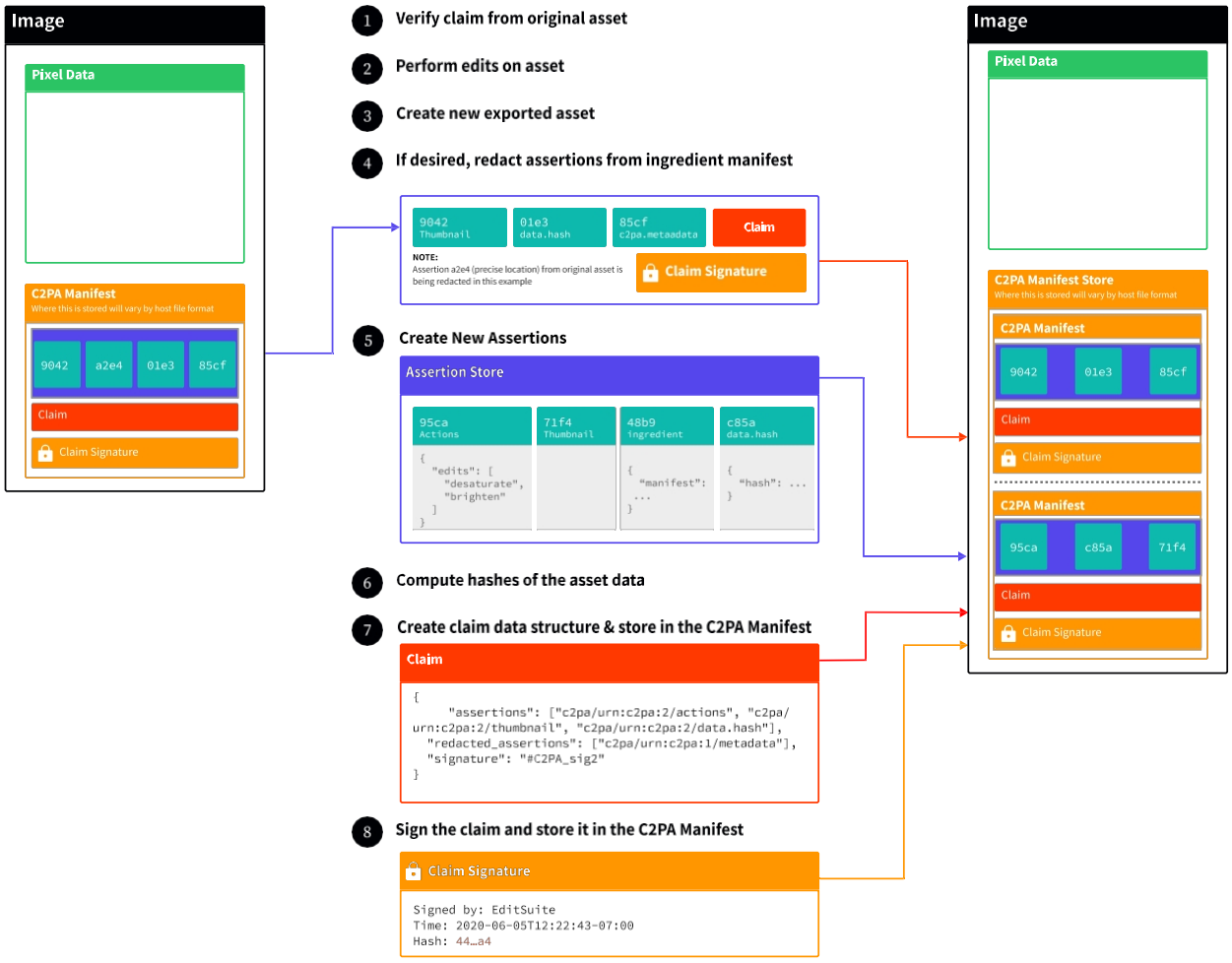

Abbildung 6. Redigieren von Behauptungen in einer sekundären Behauptung

10.4. Mehrstufige Verarbeitung

Bei einigen Asset-Dateiformaten müssen die Datei-Offsets des C2PA Manifest Store und des Asset-Inhalts vor der Signierung des Manifests festgelegt werden, damit die Inhaltsbindungen korrekt mit den von ihnen authentifizierten Inhalten übereinstimmen. Leider kann die Größe eines Manifests und seiner Signatur erst nach der Signierung genau bestimmt werden, was zu Änderungen der Datei-Offsets führen kann.

In JPEG 1-Dateien muss beispielsweise der gesamte C2PA-Manifest-Speicher vor den Bilddaten in der Datei erscheinen, sodass seine Größe die Datei-Offsets der zu authentifizierenden Inhalte beeinflusst.

Um dies zu erreichen, wird ein mehrstufiger Ansatz verfolgt, ähnlich wie bei Signaturen in PDF-Dateien.

10.4.1. Erstellen von Inhaltsbindungen

Bei der Erstellung eines Standardmanifests muss dessen Anspruch eine oder mehrere Inhaltsbindungsaussagen in seiner Liste von Aussagen enthalten, um sicherzustellen, dass die Datei manipulationssicher ist.

Erstellen Sie die Daten-Hash-Assertion und fügen Sie sie unter Berücksichtigung der folgenden Überlegungen zum Assertion-Speicher hinzu.

In vielen Fällen, wie beispielsweise bei JPEG 1, ist es nicht möglich, das Asset in seiner Gesamtheit zu hashen, da das Manifest in der Mitte der Datei eingebettet ist und daher die Größe oder Position der Manifestdaten zum Zeitpunkt der Berechnung des Asset-Hashs nicht bekannt ist. Diese zirkuläre Abhängigkeit wird vermieden, indem beim Hashing Ausschlussbereiche angegeben werden können. Wenn Ausschlussbereiche angegeben werden, wird ein einziger Hash durchgeführt, jedoch nur über die Asset-Bereiche, die nicht in einem der Ausschlüsse enthalten sind.

Wenn ein Manifest in der Mitte einer JPEG 1-Datei in einem APP11-Segment eingebettet ist, kann der Ersteller der Forderung das/die APP11-Segment(e) aus der Hash-Berechnung ausschließen.

Um Insertionsangriffe zu verhindern, ist es wünschenswert, nach Möglichkeit nur einen einzigen Ausschlussbereich zu haben. Wenn die Größe oder Position (oder beides) des Manifests in der Ressource nicht bekannt ist, müssen die `Start`- und Längenwerte in der Daten-Hash-Assertion beide Null sein, und die Größe des Pad-Werts sollte groß genug sein, um das Schreiben der Werte während des zweiten Durchgangs zu ermöglichen. Es werden mindestens 16 Byte empfohlen. Der Wert des `Pad`-Schlüssels muss aus lauter 0x00 bestehen.

Wenn eine Auffüllung verwendet wird, ist es möglich, dass die Auffülldaten geändert werden können, ohne dass dies zu einem Validierungsfehler führt. Anspruchgeneratoren müssen sicherstellen, dass Änderungen an den Auffülldaten (oder anderen ausgeschlossenen Asset-Daten) die Interpretation des Assets nicht verändern können.

HINWEIS

Bei JPEG 1-Dateien kann dies entweder durch Eliminieren der Auffüllung oder durch Sicherstellen erreicht werden, dass die `JFIF APP11/C2PA`-Segmente nicht gekürzt oder in einen anderen Segmenttyp geändert werden können. Dies ist
Dies wird erreicht, indem alle C2PA-Manifest-Segment-Header (APP11) und 2-Byte-Längenfelder in die Daten-Hash-Map für alle Manifest-haltigen Segmente aufgenommen werden. Dadurch wird sichergestellt, dass Daten, die im Ausschlussbereich geändert wurden, von JPEG-Prozessoren nicht falsch interpretiert werden.

10.4.2. Erstellen Sie einen temporären Anspruch und eine Signatur

Fügen Sie die neu erstellte Daten-Hash-Assertion-Referenz zur Assertion-Liste des Anspruchs hinzu und geben Sie einen temporären Hash-Wert an, z. B. Leerzeichen.

Zu diesem Zeitpunkt ist der temporäre Anspruch vollständig und kann zum erstellten C2PA-Manifest hinzugefügt werden.

Da es sich derzeit nur um einen vorübergehenden Anspruch handelt, ist es nicht möglich, ihn zu unterzeichnen. Um sicherzustellen, dass das Feld für die Anspruchsignatur eine gültige CBOR-Struktur enthält, erstellen Sie eine temporäre COSE_Sign1_Tagged-Struktur, wie in RFC 8152, Abschnitt 4.2 beschrieben. `COSE_Sign1_Tagged` ist ein Tag-Byte, gefolgt von einer COSE_Sign1-Struktur, bei der es sich um ein CBOR-Array mit vier Elementen handelt. Konstruieren Sie das Array wie folgt:

- Das erste Element ist der `geschützte` Header-Bucket (RFC 8152, Abschnitt 3). Erstellen Sie einen leeren Bucket, indem Sie an dieser Stelle ein `bstr` der Größe 0 an dieser Position ein.
- Das zweite Element ist der `ungeschützte` Header-Bucket, bei dem es sich um eine CBOR-Map handelt. Erstellen Sie eine Map mit einem Paar. Verwenden Sie die Zeichenfolge `„pad“` als Bezeichnung und platzieren Sie einen `bstr` mit der gewünschten Auffüllgröße, gefüllt mit Null-Bytes (0x00), als Wert. Für die Anfangsgröße dieser Auffüllung wird eine Größe von 25 Kilobyte empfohlen.
- Das dritte Element ist die `Nutzlast`. Setzen Sie hier den Wert `nil` (CBOR-Haupttyp 7, Wert 22).

- Das vierte Element ist `die Signatur`. Platzieren Sie hier einen `bstr` der Größe 0.

10.4.3. Vervollständigen Sie das C2PA-Manifest

Zu diesem Zeitpunkt sind alle Felder, aus denen sich das gesamte C2PA-Manifest für das Asset zusammensetzt, ausgefüllt und können (sofern noch nicht geschehen) in ihre endgültige Form gebracht werden. Das C2PA-Manifest des Assets wird zusammen mit den Manifesten aller Bestandteile zu einem vollständigen C2PA-Manifest-Speicher zusammengefasst. Das aktive Manifest muss das letzte C2PA-Manifest-Superbox im C2PA-Manifest-Speicher-Superbox sein. Der C2PA-Manifest-Speicher kann dann wie in Abschnitt 11.3, „Einbetten von Manifesten in verschiedene Dateiformate“, beschrieben in das Asset eingebettet werden.

10.4.4. Zurückgehen und ausfüllen

Nachdem der C2PA Manifest Store in das Asset eingebettet wurde, können der Start-Offset und die Länge des aktiven Manifests in seiner Daten-Hash-Assertion aktualisiert werden. Dabei ist es wichtig, dass Sie nicht die Größe des Assertion-Feldes ändern, sondern nur dessen Daten. Dies geschieht durch Anpassen des Werts des Pad-Feldes auf die erforderliche Länge, um die verbleibenden Bytes „aufzufüllen”.

HINWEIS

Die bevorzugte/deterministische CBOR-Serialisierung von `Pad` verwendet eine Integer-Variablenlänge, um die Länge der codierten Binärdaten anzugeben. Wenn die Länge von Null auf 1 Byte oder von 1 auf 2 Byte (usw.) wechselt, springt die Länge des resultierenden Pads um zwei Bytes. Das bedeutet, dass nicht alle Paddings mit einem Einzelnes Auffüllfeld. Beispielsweise können 24-Byte- und 26-Byte-Auffüllungen erstellt werden, eine 25-Byte-Auffüllung jedoch nicht. In diesem Fall kann die gewünschte Auffüllung zwischen `Pad` und `Pad2` aufgeteilt werden. Um beispielsweise ein 25-Byte-Pad zu erstellen, kann ein Anspruchgenerator 19 Bytes in `Pad` (was zu einer codierten Länge von 20 Bytes führt) und 4 Bytes in `Pad2` (was zu 5 Bytes führt) codieren.

Sobald die Daten-Hash-Assertion aktualisiert wurde, kann sie gehasht werden und der Hash kann über die leeren Stellen geschrieben werden, die zuvor zur Speicherung des Speicherorts verwendet wurden.

Der Anspruch ist nun vollständig und kann wie in Abschnitt 10.3.2.4, „Signieren eines Anspruchs“, beschrieben gehasht und signiert werden, wobei die resultierende Signatur den vorab zugewiesenen Speicherplatz ausfüllt. Der Pad-Header kann dann nach Bedarf verkleinert werden, sodass das Feld für die Anspruchsignatur dieselbe Größe behält. Da dieser Header ungeschützt ist, führt seine Änderung nicht zur Ungültigkeit der Anspruchsignatur.

Wenn die serialisierte Struktur `COSE_Sign1_Tagged` die reservierte Größe des Feldes „C2PA Claim Signature“ überschreitet, muss die mehrstufige Verarbeitung mit einer größeren Auffüllgröße wiederholt werden, die in Abschnitt 10.4.2, „Erstellen eines temporären Anspruchs und einer Signatur“, ausgewählt wird. Die beim vorherigen Versuch abgerufenen Widerrufsinformationen sollten wiederverwendbar sein, wenn sie sich noch innerhalb ihres Gültigkeitsintervalls befinden (RFC 6960, Abschnitt 4.2.2.1), aber für den neuen Anspruch ist ein neuer Zeitstempel erforderlich, da sich die Datei-Offsets aufgrund der hinzugefügten Auffüllung geändert haben.

Ein C2PA-Manifest kann außerhalb dieser Spezifikation definierte Assertions enthalten, die vom Dateilayout abhängen können. Daher ist es möglich, dass der Anspruchgenerator das Dateilayout und/oder die Offsets in einer Daten-Hash-Assertion nicht mehr ändern kann. In diesem Fall sollten Anspruchgeneratoren vor der Erstellung der Assertion eine Auffüllung verwenden, um sicherzustellen, dass das Dateilayout nach Fertigstellung der Assertion nicht mehr geändert werden muss.

Kapitel 11. Manifeste

11.1. Verwendung von JUMBF

11.1.1. Begründung

Um viele der Anforderungen von C2PA zu erfüllen, mussten C2PA-Manifeste in einem strukturierten Binärdatenspeicher gespeichert (serialisiert) werden, der bestimmte Funktionen ermöglicht, darunter:

- Möglichkeit, mehrere Manifeste (z. B. Eltern und Inhaltsstoffe) in einem einzigen Container zu speichern.
- Möglichkeit, über URIs auf einzelne Elemente (sowohl innerhalb als auch zwischen Manifesten) zu verweisen.
- Möglichkeit, die Teile eines Elements, die gehasht werden sollen, eindeutig zu identifizieren.
- Möglichkeit, vordefinierte Datentypen zu speichern, die von C2PA verwendet werden (z. B. JSON und CBOR).
- Möglichkeit, beliebige Datenformate (z. B. XML, JPEG usw.) zu speichern.

Unser ausgewähltes Containerformat – ISO 19566-5:2023 (JUMBF) – unterstützt nicht nur alle oben genannten Anforderungen, sondern wird auch nativ von der JPEG-Formatfamilie unterstützt und ist mit dem boxbasierten Modell (d. h. ISOBMFF, ISO 14496-12) kompatibel, das von vielen gängigen Bild- und Videodateiformaten verwendet wird. Die Verwendung von JUMBF bietet dieselben Vorteile (und einige zusätzliche, wie z. B. URI-Referenzen) und ermöglicht gleichzeitig die Arbeit mit klassischen Bildformaten wie JPEG/JFIF und PNG sowie 3D- und Dokumentformaten (z. B. PDF). Dieses serialisierte Format sollte auch in Formaten verwendet werden, die JUMBF nicht nativ unterstützen, oder wenn C2PA-Manifest-Speicher separat vom Asset gespeichert werden, z. B. in einer separaten Datei oder an einem separaten URI-Speicherort.

HINWEIS

Da die meisten Standardaussagen sowie die Anspruchsignatur als CBOR serialisiert sind, wurde die Verwendung von CBOR für das gesamte C2PA-Manifest in Betracht gezogen, aber nicht gewählt, da CBOR kein Containerformat ist.

Um beispielsweise einen „JSON-Blob“ in CBOR zu speichern und sicherzustellen, dass es sich um JSON (und nicht um ein anderes Format) handelt, müsste eine Datenstruktur für die Speicherung solcher Elemente entworfen werden. Anschließend müsste die übergeordnete Struktur definiert werden, um festzulegen, wie diese Struktur übertragen werden soll. Das gleiche Konzept müsste auch für jede der nativen Funktionen von JUMBF angewendet werden.

Es wäre zwar durchaus möglich, alle erforderlichen Funktionen vollständig in CBOR neu zu implementieren, dies wäre jedoch mit einem hohen Arbeitsaufwand verbunden und würde die Notwendigkeit eines JUMBF/BMFF-Parsers in allen Implementierungen nicht vollständig beseitigen.

11.1.2. Verarbeitungsregeln

Ein C2PA-Manifest-Verbraucher darf niemals eine Assertion, einen Assertion-Speicher, einen Claim, eine Claim-Signatur oder ein C2PA-Manifest verarbeiten, die nicht in einem C2PA-Manifest-Speicher enthalten sind. Wenn ein C2PA-Manifest-Verbraucher auf eine JUMBF-Box oder Superbox stößt, deren JUMBF-Typ-UUID er nicht erkennt, muss er deren Inhalt überspringen (und ignorieren).

HINWEIS	Das bedeutet, dass der C2PA-Manifest-Verbraucher private Boxen verarbeiten kann, die ihm bekannt sind, aber solche ignorieren muss, die ihm nicht bekannt sind.

Wenn die Schaltflächen *„Requestable“ (Anforderbar)* und *„Label Present“ (Etikett vorhanden)* im Feld „JUMBF Description“ (JUMBF-Beschreibung) eines beliebigen JUMBF-Feldes oder Superfeldes aktiviert sind, muss dieses Feld oder Superfeld in jedem aktualisierten C2PA Manifest Store beibehalten werden.

HINWEIS	Boxen, bei denen diese Schalter aktiviert sind, sollen über JUMBF-URIs referenziert werden, und ihre Entfernung könnte zu Fehlern in nachgelagerten Workflows führen.

11.1.3. Erweiterungen

11.1.3.1. Allgemeines

In diesem Abschnitt werden die für diese Spezifikation erforderlichen Erweiterungen der JUMBF-Spezifikation (ISO 19566-5:2023) beschrieben.

11.1.3.2. Komprimierte Boxen

Um die Komprimierung von Manifesten zu unterstützen, wird von C2PA eine neue Brob-Inhaltsbox unterstützt. Basierend auf einer ähnlichen Box in JPEG-XL (ISO/IEC 18181-2:2024) ist die Brob-Box eine Inhaltsbox, deren Inhalt aus den Brotli-komprimierten Bytes entweder eines Standardmanifests oder eines Aktualisierungsmanifests besteht, wie in der Klausel zu komprimierten Manifesten beschrieben. Die `Brob`-Box muss die Box-ID `0x62726F62` (`brob`) haben.

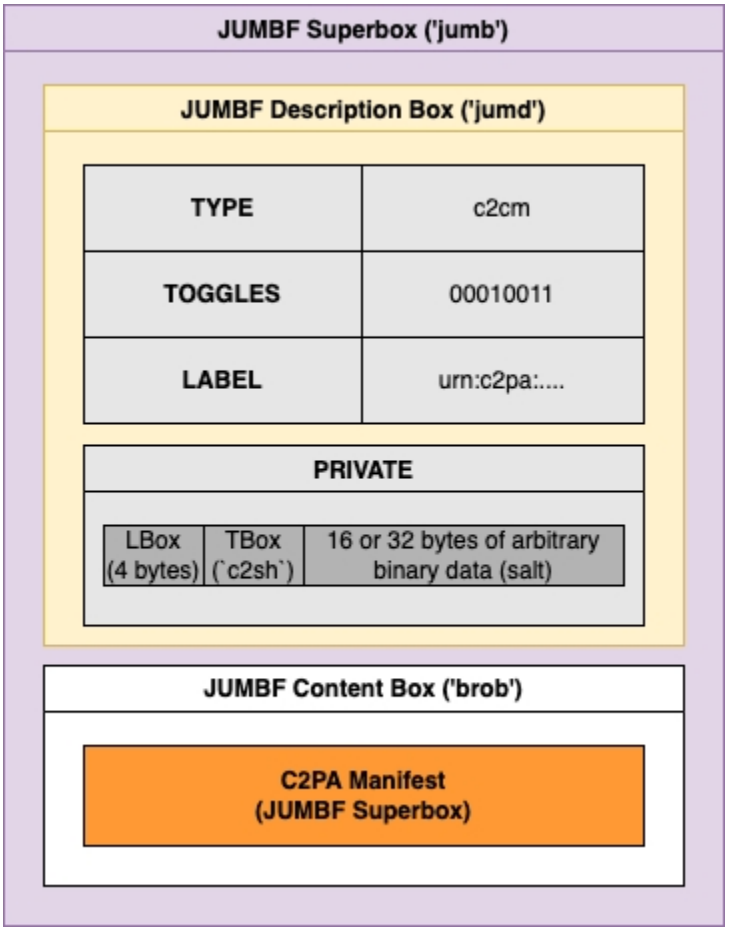

Abbildung 7. Beispiel für ein komprimiertes Manifest

Das Hashing einer komprimierten Box erfolgt auf dieselbe Weise wie bei jeder anderen Box, wie in Abschnitt 8.4.2.3, „Hashing von JUMBF-Boxen“, beschrieben.

HINWEIS

Dies bedeutet, dass bei einer hashed_uri-Referenz aus einer Ingredient Assertion zu einem C2PA-Manifest über das Feld `activeManifest` der Hash-Wert nach dem gleichen Verfahren wie bei jeder anderen JUMBF-Superbox berechnet wird: über die JUMBF-Beschreibungsbox und die Brob-Box mit ihrer komprimierten Nutzlast, jedoch ohne den Header der Superbox. Der Inhalt der Brob-Box wird zur Berechnung des Hash-Werts nicht zuvor dekomprimiert.

11.1.4. C2PA-Box-Details

11.1.4.1. JUMBF-Beschreibungsboxen

11.1.4.1.1. Labels

Wie in der JUMBF-Spezifikation (ISO 19566-5:2023, A.3) beschrieben, muss ein Label als ISO/IEC 10646-Zeichen in der UTF-8-Kodierung gespeichert werden. Zeichen im Bereich von U+0000 bis einschließlich U+001F und von U+007F bis einschließlich U+009F sowie die spezifischen Zeichen „/“, „:“, „?“ und „#“ sind in der Kennzeichnung nicht zulässig. Die Kennzeichnung muss null-terminiert sein.

Als Teil von JUMBF-URIs dürfen die Zeichen U+FEFF, U+FFFF und U+D800-U+DFFF ebenfalls nicht verwendet werden.

11.1.4.1.2. Umschalter

Alle JUMBF-Beschreibungsfelder (ISO 19566-5:2023, A.3), die in einem C2PA-Manifest verwendet werden, erfordern ein Label, das *Label Present*-Toggle (`xxxxxx1x`) muss gesetzt werden. Da JUMBF-URIs außerdem dazu verwendet werden, um auf Felder im gesamten System zu verweisen (z. B. Auflistung von Angaben, Verweise auf Inhaltsstoffe usw.), muss der Schalter *„Requestable“* (`xxxxxx11`) gesetzt werden.

Wenn ein Salt in eine PRIVATE-Box gemäß Abschnitt 8.4.2.3, „Hashing JUMBF Boxes“ (Hash-Funktion für JUMBF-Boxen), aufgenommen wird, muss auch der Schalter *„Private“* (`xxx1xxxx`) gesetzt werden.

11.1.4.2. Manifest-Speicher

C2PA-Daten werden in eine JUMBF-kompatible Box-Struktur serialisiert. Die äußerste Box wird als C2PA Manifest Store bezeichnet, auch bekannt als Content Credentials. Abbildung 8, „C2PA Manifest Store“, ist ein Beispiel für einen C2PA Manifest Store mit einem einzelnen C2PA Manifest:

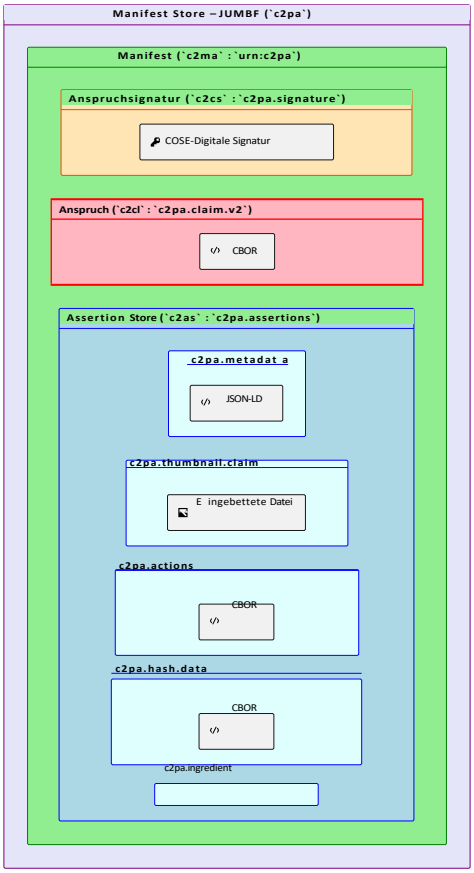

Abbildung 8. C2PA Manifest Store

Der C2PA Manifest Store ist eine JUMBF-Superbox, die aus einer Reihe anderer JUMBF-Boxen und Superboxen besteht, die jeweils durch ihre eigene JUMBF-Typ-UUID und ihre Bezeichnung in ihrer JUMBF-Beschreibungsbox identifiziert werden. Der C2PA Manifest Store muss die Bezeichnung `c2pa` und die JUMBF-Typ-UUID `63327061-0011-0010-8000-00AA00389B71` (`c2pa`) haben und eine oder mehrere C2PA Manifest Superboxen, auch bekannt als C2PA Manifests, enthalten. Der C2PA-Manifest-Speicher kann auch JUMBF-Felder und Superboxen enthalten, deren JUMBF-Typ-UUIDs in dieser Spezifikation nicht definiert sind.

HINWEIS Durch die Zulassung anderer Boxen und Superboxen werden benutzerdefinierte Erweiterungen von C2PA sowie das Hinzufügen neuer Boxen in zukünftigen Versionen dieser Spezifikation ermöglicht, ohne die Kompatibilität zu beeinträchtigen.

Jedes C2PA-Manifest muss die zum Zeitpunkt der Ausstellung eines Anspruchs erstellten Daten enthalten, darunter den C2PA-Assertion-Store, einen C2PA-Anspruch und eine C2PA-Anspruchsignatur. Ein C2PA-Manifest kann auch JUMBF-Boxen und Superboxen enthalten, deren JUMBF-Typ-UUIDs in dieser Spezifikation nicht definiert sind.

Die UUID vom Typ JUMBF für jedes C2PA-Manifest muss entweder `63326D61-0011-0010-8000-00AA00389B71` (`c2ma`) oder `6332756D-0011-0010-8000-00AA00389B71` (`c2um`) je nach Art des Manifests. Das Feld „C2PA Manifest“ muss mit einem `urn:c2pa`-Wert beschriftet werden, der wie unter „Eindeutige Identifikatoren” beschrieben berechnet wird.

11.1.4.3. Assertion Store

Der C2PA-Assertion-Speicher ist eine Superbox, die die Bezeichnung `c2pa.assertions` und eine UUID vom Typ JUMBF mit dem Wert `63326173-0011-0010-8000-00AA00389B71` (`c2as`) haben. Er muss eine oder mehrere JUMBF-Superboxen (genannt

C2PA-Assertion-Boxen), deren JUMBF-Typ den Typ der Unterboxen definiert, die die Assertion-Daten enthalten (ISO 19566-5:2023, Anhang B). Diese Superboxen müssen jeweils eine Kennzeichnung gemäß der Definition in Standard Assertions aufweisen und eine JUMBF-Beschreibungsbox, eine oder mehrere JUMBF-Inhaltsboxen und möglicherweise eine Auffüllbox enthalten (ISO 19566-5:2023, A.4).

Der JUMBF-Inhaltstyp (ISO 19566-5:2023, Anhang B) in jeder Assertion-Superbox sollte vom Typ CBOR-Inhaltstyp (`cbor`), JSON-Inhaltstyp (`json`), eingebetteter Datei-Inhaltstyp (`bfdb` & `bidb`) oder UUID-Inhaltstyp (`uuid`) sein, wobei jedoch jeder in JUMBF (ISO 19566-5:2023) und seinen Änderungen definierte Inhaltstyp zulässig ist. Darüber hinaus kann auch eine JUMBF-Schutzbox gemäß ISO 19566-4:2020 verwendet werden.

HINWEIS	Benutzerdefinierte Assertions, die andere Formate/Serialisierungen von Daten enthalten, wie z. B. verschlüsselte Daten, sind unterstützt durch die Verwendung einer UUID-Inhaltsbox, die die benutzerdefinierte UUID gefolgt von den Daten enthält (ISO 19566-5:2023, B.5).

11.1.4.4. Anspruch und Anspruchsignatur

Die C2PA-Anspruchsbox muss die Bezeichnung `c2pa.claim.v2`, eine UUID vom Typ JUMBF mit dem Wert `6332636C-0011-0010-8000-00AA00389B71` (`c2cl`) haben und aus einer einzigen CBOR-Inhaltstypbox (`cbor`) bestehen.

Das Feld „C2PA-Anspruchsignatur“ muss die Bezeichnung „`c2pa.signature`“, eine UUID vom Typ JUMBF mit dem Wert `63326373-0011-0010-8000-00AA00389B71` (`c2cs`) haben und aus einem einzigen Feld vom Typ „CBOR-Inhaltstyp“ (`cbor`) bestehen.

11.1.4.5. Lagerung von Inhaltsstoffen

Wenn ein C2PA-Manifest Angaben zu Inhaltsstoffen enthält und ein Inhaltsstoff ein C2PA-Manifest enthält, muss dieses C2PA-Manifest beigefügt werden, um sicherzustellen, dass die Herkunftsdaten vollständig bleiben. Solche Inhaltsstoffmanifeste werden dem C2PA-Manifest-Speicher als Peer des C2PA-Manifests für das Asset selbst hinzugefügt.

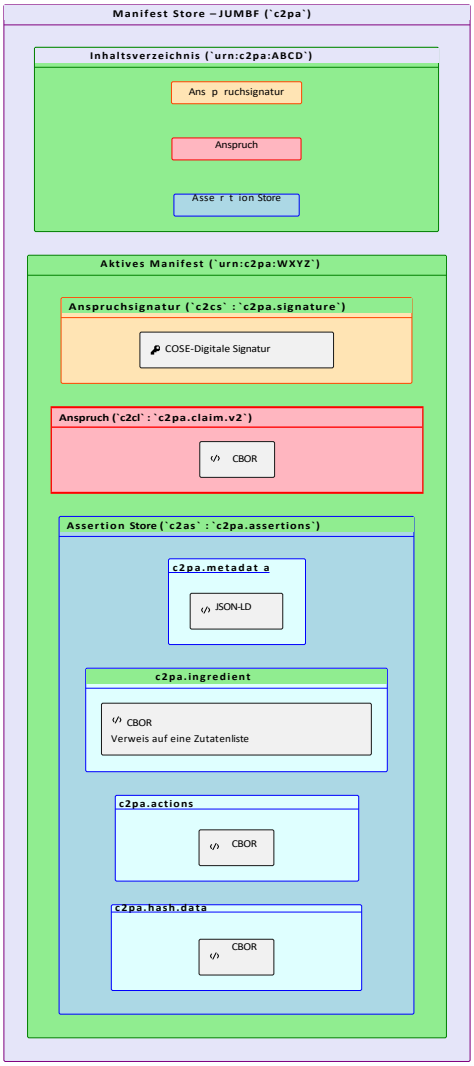

Abbildung 9. C2PA-Manifest-Speicher mit einer Zutat

11.1.4.6. Datenspeicher

WICHTIG

Dieser Abschnitt wird aus historischen Gründen beibehalten. Das Konzept einer Datenbox wurde zugunsten einer Standardaussage verworfen, die eine Standard-JUMBF-Box vom Typ „Embedded File“ zur Speicherung der Daten verwendet. Weitere Informationen zur eingebetteten Datenaussage finden Sie in Abschnitt 18.12, „Eingebettete Daten“.

Ein C2PA-Datenboxspeicher ist eine JUMBF-Superbox, die nur eine oder mehrere CBOR-Inhaltstypboxen (`cbor`) enthalten darf. Sie darf keine anderen Arten von JUMBF-Boxen oder Superboxen enthalten. Sie muss die Bezeichnung `c2pa.databoxes` und eine JUMBF-Typ-UUID von `63326462-0011-0010-8000-00AA00389B71` (`c2db`) haben.

Die CBOR-Inhaltstyp-Boxen müssen die Bezeichnung `c2pa.data` (für eingebettete Daten) haben.

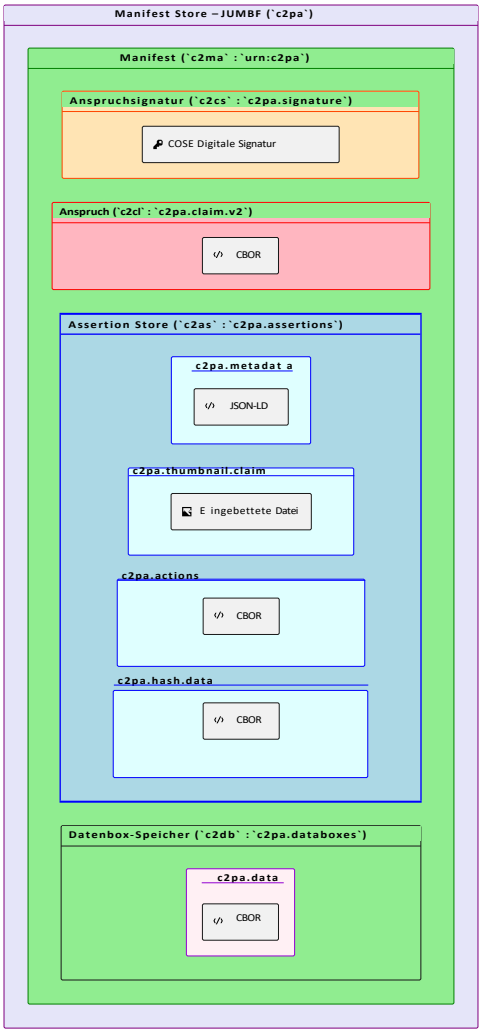

Abbildung 10. C2PA Manifest Store mit Datenboxen

11.2. Arten von Manifesten

11.2.1. Gemeinsamkeiten

Alle C2PA-Manifeste müssen einen Assertion Store mit mindestens einer Assertion, einem Claim und einer Claim-Signatur enthalten.

11.2.2. Standardmanifeste

Ein Standard-C2PA-Manifest (JUMBF-Typ UUID: `63326D61-0011-0010-8000-00AA00389B71` (`c2ma`)) muss genau eine feste Bindung an eine Inhaltsaussage enthalten – entweder eine `c2pa.hash.data`, `c2pa.hash.boxes`, `c2pa.hash.collection.data`, `c2pa.hash.bmff.v2` (veraltet) oder `c2pa.hash.bmff.v3`, je nach Art der Ressource und Version, für die das Manifest bestimmt ist. Aufgrund dieser Anforderung sind sie der vorherrschende Manifesttyp, der in C2PA-Herkunftsdaten vorhanden sein wird.

Manifest-Konsumenten müssen auch Standard-C2PA-Manifeste akzeptieren, die mit dem JUMBF-Typ UUID `63326D64-0011-0010-8000-00AA00389B71` (c2md) spezifiziert sind, aber Claim-Generatoren dürfen keine Manifeste mit diesem JUMBF-Typ UUID erstellen.

HINWEIS Ein Standard-C2PA-Manifest kann entweder als aktives Manifest oder als Inhaltsstoffmanifest vorliegen.

11.2.3. Manifeste aktualisieren

Es gibt jedoch Provenienz-Workflows, bei denen zusätzliche Assertions hinzugefügt werden müssen, ohne dass der digitale Inhalt geändert wird. In diesen Workflows sollte ein Update Manifest (JUMBF-Typ UUID: `6332756D-0011-0010-8000-00AA00389B71` (`c2um`)) verwendet werden.

Ein Update-Manifest darf keine Assertions der Typen `c2pa.hash.data`, `c2pa.hash.boxes`, `c2pa.hash.collection.data`, `c2pa.hash.bmff.v2` (veraltet) oder `c2pa.hash.bmff.v3` enthalten, da sich der Inhalt nicht geändert hat und daher die Bindungen nicht aktualisiert werden müssen. Im Falle eines Datei-Offset-Hash (`c2pa.hash.data`) muss der C2PA-Manifest-Speicher nach der Aktualisierung weiterhin mit demselben Datei-Offset beginnen – nur seine Länge darf sich ändern. Darüber hinaus darf er keine `c2pa.hash.` multi-asset-Assertion enthalten.

Ein Update-Manifest kann Assertions vom Typ `c2pa.actions` oder `c2pa.actions.v2` enthalten, vorausgesetzt, dass der Wert des Aktionsfeldes jeder Aktion, die im Aktionsarray dieser Assertions vorhanden ist, nur einer der folgenden Werte ist:

- `c2pa.edited.metadata`
- `c2pa.opened`
- `c2pa.veröffentlicht`
- `c2pa.redacted`

Ein Update-Manifest darf keine Assertion vom Typ `c2pa.actions` oder `c2pa.actions.v2` enthalten, die eine Aktionsfeld außerhalb dieser Liste.

Ein Update-Manifest kann entweder eine Zeitstempel-Assertion, eine Zertifikatsstatus-Assertion oder beides enthalten.

HINWEIS Dies ist der Ersatzansatz für die veraltete Funktion der Zeitstempel-Manifeste.

Ein Update-Manifest darf keine Miniaturansicht-Assertion enthalten.

HINWEIS Der Grund für diese Anforderungen ist, dass ein Aktionsfeld außerhalb dieser Liste oder eine Miniaturansicht Änderungen an den digitalen Inhalten impliziert.

Das Update-Manifest muss genau eine `c2pa.ingredient.`v3-Aussage enthalten, die (a) sowohl das Feld `„activeManifest“` als auch das Feld `„claimSignature“` mit Werten enthält, die die URI-Referenzen zum C2PA-Manifest bzw. zur Anspruchsignatur sind (oder ein `c2pa.ingredient.v2` oder `c2pa.ingredient`, das ein c2pa_manifest-Feld enthält) des zu aktualisierenden Assets enthält und (b) den Wert `parentOf` für das Beziehungsfeld hat.

HINWEIS Das C2PA-Manifest der Zutat kann entweder ein Standardmanifest oder ein Aktualisierungsmanifest sein.

11.2.4. Komprimierte Manifeste

Standard- und Aktualisierungsmanifeste können wie oben beschrieben vollständig mit dem Brotli-Komprimierungsalgorithmus komprimiert werden. Für beide Manifesttypen muss der Wert des Feldes `TYPE c2cm` lauten, der Wert des Feldes label muss mit dem Label der komprimierten Manifest-Superbox identisch sein, und der Inhalt des Feldes `brob` content muss aus den komprimierten Bytes der gesamten Manifest-Superbox bestehen. Ein Beispiel für ein komprimiertes Standardmanifest finden Sie in Abbildung 7, „Beispiel für ein komprimiertes Manifest“.

WICHTIG An jeder Stelle in dieser Spezifikation, an der auf ein Standard- oder Aktualisierungsmanifest verwiesen wird, ist auch ein komprimiertes Standard- oder Aktualisierungsmanifest gültig.

11.2.5. Zeitstempel-Manifeste (HISTORISCH)

WICHTIG Diese Funktion wurde zugunsten der Zeitstempel-Assertion als veraltet deklariert und darf nicht geschrieben werden. von Anspruchgeneratoren geschrieben oder von Manifestkonsumenten gelesen werden. Stattdessen wird eine Zeitstempel-Assertion verwendet, um die gleichen Ziele zu erreichen.

HINWEIS Die folgenden Informationen werden aus historischen Gründen beibehalten.

In einigen Provenienz-Workflows wird ein Standard- oder Aktualisierungsmanifest offline erstellt, wobei es nicht möglich ist, zum Zeitpunkt der Signatur einen vertrauenswürdigen Zeitstempel (gemäß RFC 3161) von einer TSA zu erhalten. Um dies zu berücksichtigen, kann ein Zeitstempel-Manifest (JUMBF-Typ UUID: `6332746D-0011-0010-8000-00AA00389B71 (c2tm)`) verwendet werden, um den Zeitstempel in einem späteren Vorgang hinzuzufügen, wenn eine TSA kontaktiert werden kann.

11.3. Einbetten von Manifesten in verschiedene Dateiformate

Ein C2PA-Manifest kann in eine Vielzahl von Dateiformaten eingebettet werden, darunter Bild-, Video-, Audio-, Schriftarten- und Dokumentdateien. Anhang A, *Einbetten von Manifesten*, enthält technische Details dazu, wie C2PA-Manifeste in jedes speziell unterstützte Dateiformat eingebettet werden können.

HINWEIS Viele klassische Bildformate wie BMP unterstützen die Einbettung beliebiger Daten nicht, sodass die Verwendung eines externen Manifests erforderlich ist.

11.4. Externe Manifeste

In einigen Fällen ist es möglicherweise nicht möglich (oder praktikabel), einen C2PA-Manifest-Speicher in eine Ressource einzubetten. In diesen Fällen ist die externe Speicherung der C2PA-Manifeste außerhalb der Ressource ein akzeptables Modell, um die Herkunft von Ressourcen nachzuweisen. Das C2PA-Manifest sollte an einem Ort gespeichert werden, der als Manifest-Repository bezeichnet wird und von einem Manifest-Verbraucher, der mit dem Asset arbeitet, leicht zu finden ist, z. B. über einen Verweis oder eine URI. Da es sich bei dem C2PA-Manifest-Speicher um eine JUMBF-Box handelt, muss er mit dem JUMBF-Medientyp `application/c2pa` bereitgestellt werden.

HINWEIS Frühere Versionen dieser Spezifikation verwendeten den Medientyp `application/x-c2pa-manifest-store` für den C2PA Manifest Store. Dieser Medientyp ist veraltet.

Einige häufige Gründe für die Verwendung eines externen Manifests sind:

- Es ist technisch möglicherweise nicht möglich, z. B. bei einer `.txt`-Datei.
- Es kann unpraktisch sein, beispielsweise wenn die Größe des C2PA Manifest Store größer ist als der digitale Inhalt des Assets.
- Es ist möglicherweise nicht angemessen, z. B. wenn dadurch ein Asset geändert würde, das nicht geändert werden sollte.

> **HINWEIS** Ein gutes Beispiel hierfür ist die Erstellung eines Manifests für ein bereits vorhandenes Asset.

11.5. Einbetten eines Verweises auf ein externes Manifest

Wenn das Asset eingebettete XMP-Daten enthält und das C2PA-Manifest extern gespeichert wird, wird empfohlen, dass der Claim-Generator einen `dcterms:provenance`-Schlüssel zum XMP hinzufügt, dessen Wert (eine URI-Referenz) angibt, wo sich das aktive Manifest befindet.

> **HINWEIS** In einer früheren Version dieser Spezifikation wurde diese Methode auch für Verweise auf eingebettete Manifeste empfohlen. Jetzt gilt dieser Mechanismus nur noch für externe Manifeste.

Da Schriftarten XMP nicht unterstützen, wird in diesem Abschnitt über Schriftarten eine gleichwertige Methode zur Angabe einer URI zu einem entfernten C2PA-Manifest-Speicher beschrieben.

Kapitel 12. Entitätsdiagramm

Abbildung 11, „C2PA-Entitätsdiagramm“, zeigt, wie alle Teile des C2PA-Systems miteinander integriert sind und in Beziehung zueinander stehen.

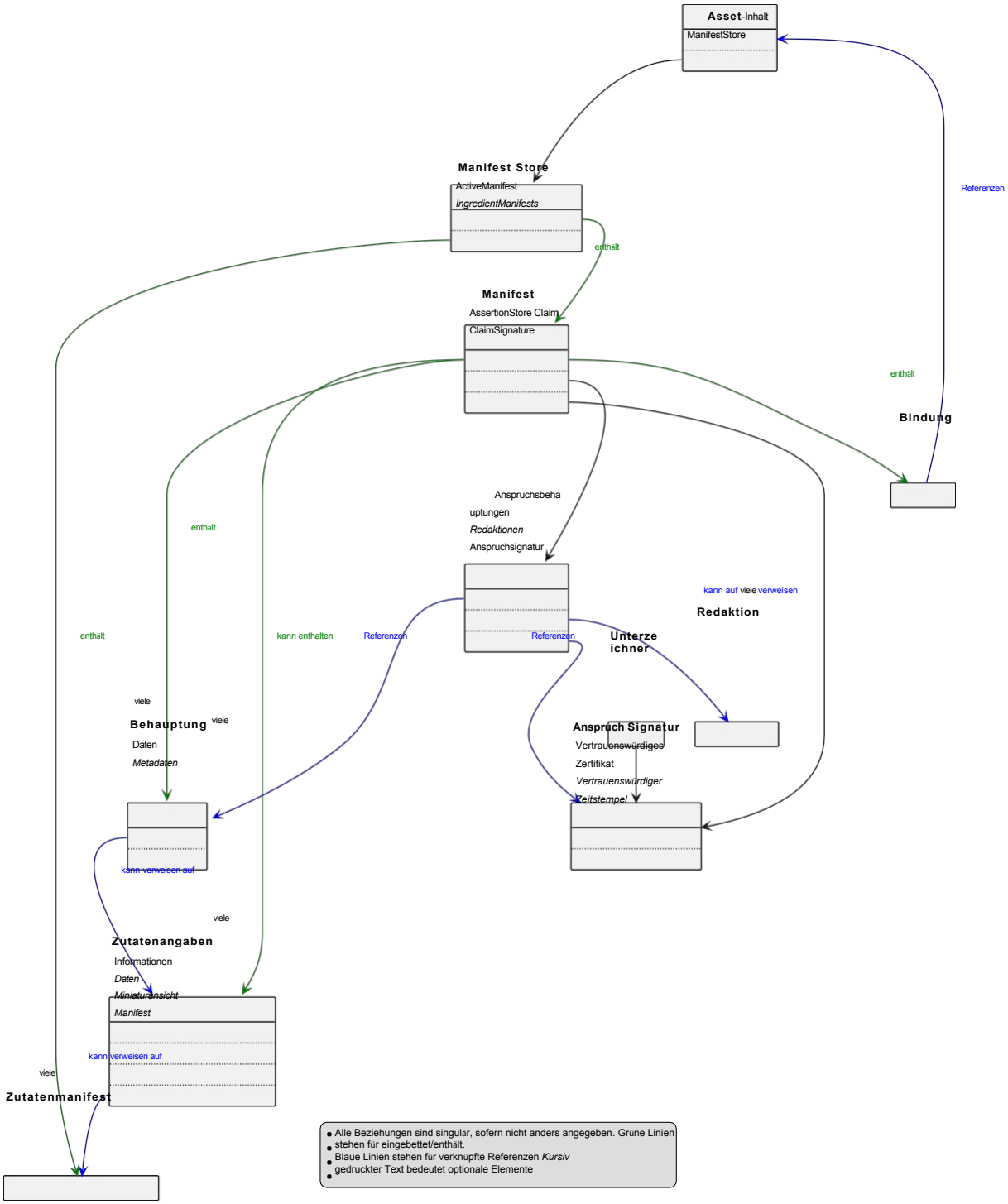

Abbildung 11. C2PA-Entitätsdiagramm

Kapitel 13. Kryptografie

13.1. Hashing

Alle kryptografischen Hashes, die gemäß den technischen Anforderungen dieser Spezifikation angewendet werden, müssen unter Verwendung eines der in diesem Abschnitt beschriebenen Hash-Algorithmen generiert werden. In diesem Abschnitt werden beide definiert:

- Eine Liste von Hash-Algorithmen, die für die Generierung von Hashes neuer Inhalte zulässig sind und für die Validierung von Hashes bestehender Inhalte erforderlich sind (die Liste der zulässigen Algorithmen);
- Eine Liste von Hash-Algorithmen, die für die Validierung von Hashes bestehender Inhalte unterstützt werden müssen, aber nicht für die Generierung von Hashes neuer Inhalte zulässig sind (die Liste der veralteten Algorithmen).

HINWEIS Dieser Abschnitt gilt nicht für Algorithmen, die für Soft-Bindings gemäß Abschnitt 18.10, „Soft-Binding“, verwendet werden.

HINWEIS Dieser Abschnitt gilt nicht für Algorithmen, die von benutzerdefinierten Assertions verwendet werden, die außerhalb dieser Spezifikation definiert sind.

Ein Algorithmus darf in höchstens einer Liste aufgeführt sein. Ein Algorithmus, der über mehrere Ausgabelängen instanziiert wird (z. B. die verschiedenen Längen von SHA2), wird als unterschiedlicher Algorithmus betrachtet, und jede Instanziierung muss separat aufgeführt werden. Wenn ein Algorithmus in keiner der beiden Listen aufgeführt ist, ist er verboten und darf nicht verwendet oder unterstützt werden. Algorithmen können aus den Listen entfernt werden, um ein Verbot eines Algorithmus zu implementieren. Aus diesem Grund dürfen Implementierungen keine zusätzlichen Algorithmen auf optionaler Basis unterstützen.

Implementierer sollten diesen Abschnitt in der aktuellen Version der Spezifikation konsultieren, wenn sie Software-Updates veröffentlichen, und sicherstellen, dass die von ihnen unterstützten Algorithmen damit konform sind.

Diese Listen legen die zulässigen Algorithmen für die Erstellung von Hashes und einen String-Algorithmus-Identifikator fest, der als Algorithmus-Identifikator (in der Regel als `„alg“` bezeichnet) im entsprechenden Feld der C2PA-Datenstrukturen verwendet wird. Die Ausgaben von Hash-Funktionen müssen als ihre in CBOR als Byte-Strings (Haupttyp 2) mit einer deklarierten Länge kodierten Binärwerte gespeichert werden. Wenn ein Feld die Ausgabe einer Hash-Funktion enthält, muss ein Algorithmus-Kennzeichnungszeichenfolgenfeld innerhalb derselben Struktur, innerhalb einer umschließenden Struktur oder in der `Claim-Map-` oder Claim-Map-v2-Struktur vorhanden sein, um anzugeben, welcher Algorithmus verwendet wurde. Ein Hash-Algorithmus-Kennzeichnungsfeld sollte genau an einer dieser Stellen vorhanden sein. Wenn jedoch mehr als eines innerhalb der Struktur und ihrer umschließenden Strukturen vorhanden ist, muss das nächstgelegene Kennzeichnungsfeld verwendet werden. Als nächstgelegen gilt zunächst ein Identifikator, der ein gleichrangiges Feld des Hashwerts ist, und dann die unmittelbar umschließende Struktur bis hin zur Stammstruktur. Ist an keiner dieser Stellen ein Identifikator vorhanden, ist das Feld `„alg“` aus der Struktur `„claim-map“` oder `„claim-map-v2“` zu verwenden.

Die zulässige Liste lautet:

- SHA2-256 („sha256”);
- SHA2-384 („sha384”);
- SHA2-512 („sha512“).

HINWEIS	Die Hash-Algorithmen der SHA-3-Familie sind aus Gründen der Konsistenz mit der Liste der zulässigen Algorithmen für digitale Signaturen nicht in der Liste der zulässigen Algorithmen enthalten, da COSE noch keine Algorithmen für digitale Signaturen festgelegt hat, die einen SHA-3-Algorithmus als Hash-Algorithmus verwenden.

Die Liste der veralteten Algorithmen ist leer.

13.2. Digitale Signaturen

Alle digitalen Signaturen, die gemäß den technischen Anforderungen dieser Spezifikation angewendet werden, müssen unter Verwendung eines der in diesem Abschnitt beschriebenen digitalen Signaturalgorithmen und Schlüsseltypen generiert werden. In diesem Abschnitt werden beide definiert:

- Eine Liste der Algorithmen für digitale Signaturen und Schlüsseltypen, die für die Generierung von Signaturen für neue Anspruchssignaturen zulässig sind und für die Validierung bestehender Anspruchssignaturen erforderlich sind (die Liste der zulässigen Algorithmen und Schlüsseltypen);
- Eine Liste der Algorithmen für digitale Signaturen und Schlüsseltypen, die für die Validierung bestehender Anspruchsignaturen unterstützt werden müssen, aber nicht für die Generierung neuer Anspruchsignaturen zulässig sind (die Liste der veralteten Algorithmen).

HINWEIS	Dieser Abschnitt regelt nicht digitale Signaturen, die von benutzerdefinierten Assertions verwendet werden, die außerhalb dieser Spezifikation definiert sind.

Diese Listen legen die zulässigen Algorithmen und Schlüsseltypen fest, indem sie auf eine Algorithmuskennung aus den relevanten Standards verweisen, die Algorithmen für COSE und deren Zuordnungen zu CBOR-Kennungen definieren, einschließlich, aber nicht beschränkt auf RFC 8152 und RFC 8230. Diese Standards legen auch den im Signaturschema verwendeten Hash-Algorithmus fest. Keine Bestimmung in Abschnitt 13.1, „Hashing“, gilt für diese Verwendung von Hash-Algorithmen; wenn ein digitaler Signaturalgorithmus in dem unten aufgeführten digitalen Signaturalgorithmus und Schlüsseltyp vorhanden ist, ist die Verwendung seines angegebenen Hash-Algorithmus im Signaturschema zulässig und muss befolgt werden.

HINWEIS	Die Erläuterungen in Klammern in den folgenden Listen dienen lediglich als Hilfe für den Leser.

13.2.1. Signaturalgorithmen

Die zulässige Liste lautet:

- ES256 (ECDSA mit SHA-256);
- ES384 (ECDSA mit SHA-384);
- ES512 (ECDSA mit SHA-512);
- PS256 (RSASSA-PSS mit SHA-256 und MGF1 mit SHA-256);
- PS384 (RSASSA-PSS mit SHA-384 und MGF1 mit SHA-384);
- PS512 (RSASSA-PSS mit SHA-512 und MGF1 mit SHA-512);
- EdDSA (Edwards-Curve DSA).
    - Nur Ed25519-Instanz. Keine anderen EdDSA-Instanzen sind zulässig.

Die Liste der veralteten Funktionen ist leer.

Implementierungen müssen überprüfen, ob die für Signatur- oder Verifizierungsvorgänge bereitgestellten Schlüssel für den gewählten Algorithmus korrekt sind, wie in RFC 8152, Abschnitt 8.1 für ECDSA, RFC 8152, Abschnitt 8.2 für EdDSA und RFC 8230, Abschnitt 2 und Abschnitt 4 für RSASSA-PSS vorgeschrieben.

Diese Anforderungen sind der Einfachheit halber hier zusammengefasst:

- ECDSA erfordert elliptische Kurvenschlüssel auf den elliptischen Kurven P-256, P-384 oder P-521.
    - Obwohl empfohlen wird, P-256-Schlüssel mit `ES256`, P-384-Schlüssel mit `ES384` und P-521-Schlüssel mit `ES512` zu verwenden, ist dies nicht zwingend erforderlich. Implementierungen müssen Schlüssel auf jeder dieser Kurven für alle ECDSA-Algorithmusauswahlen akzeptieren.
- Ed25519 erfordert elliptische Kurvenschlüssel auf der elliptischen Kurve edwards25519.
- RSASSA-PSS erfordert RSA-Schlüssel mit einer Modullänge von mindestens 2048 Bit.

Implementierungen dürfen keine Signaturen mit Schlüsseln generieren oder verifizieren, die für die gewählte Algorithmusauswahl nicht korrekt sind. Implementierungen können RSA-Schlüssel mit einer Modullänge von mehr als 16384 Bit ablehnen.

13.2.2. Verwendung von COSE

Die Signatur für die CBOR-codierte Forderung wird durch CBOR Object Signing and Encryption (COSE) gemäß RFC 8152, Abschnitte 4.2 und 4.4, erstellt.

HINWEIS

Nutzlasten können entweder innerhalb einer COSE-Signatur vorhanden sein oder separat transportiert werden („detached Inhalt" gemäß der Beschreibung in RFC 8152, Abschnitt 4.1). Im Modus „getrennter Inhalt" werden die signierten Daten außerhalb der `COSE_Sign1_Tagged`-Struktur gespeichert, und das Nutzdatenfeld der `COSE_Sign1_Tagged`-Struktur ist immer `null`.

Unabhängig davon, ob die Nutzlast in der COSE_Sign1_Tagged-Signatur enthalten oder von dieser getrennt ist, muss der Inhalt des Nutzlastfelds der `Sig_Struktur` im Speicher, wenn es zur Berechnung oder Überprüfung einer digitalen Signatur erstellt wird, mit den externen Daten gefüllt werden, wie sie durch die jeweilige Verwendung der digitalen Signatur in dieser Spezifikation beschrieben sind. Das Nutzlastfeld der `Sig_Struktur` darf niemals `nil` sein.

Bei der Berechnung oder Überprüfung der Signatur eines Standard- oder Aktualisierungsmanifests enthält das Payload-Feld der `Sig_structure` den Inhalt des JUMBF-Feldes „Claim", wie in Abschnitt 10.3.2.4, „Signieren eines Claims", und Abschnitt 11.1, „Verwendung von JUMBF", beschrieben.

13.2.3. Berechnung der Signatur

Die Signatur wird gemäß RFC 8152, Abschnitt 4.4 berechnet oder überprüft. Für die Konstruktion der `Sig_Struktur` gelten die folgenden zusätzlichen Anforderungen:

- Der Wert für das Kontextelement muss `Signature1` sein, es sei denn, eine bestimmte Verwendung digitaler Signaturen in dieser Spezifikation schreibt stattdessen die Verwendung `von CounterSignature` vor. `Signature` darf nicht verwendet werden.
- Der Wert für das Payload-Element wird durch jede Verwendung digitaler Signaturen in dieser Spezifikation festgelegt.

- Das external_aad-Element muss ein `bstr` der Länge Null sein. Extern authentifizierte Daten dürfen nicht verwendet werden.
- Der `Alg`-Header, der den Signaturalgorithmus angibt, muss im Element `body_protected` gemäß der Definition in RFC 8152, Abschnitt 3.1, vorhanden sein.

HINWEIS	Der `alg`-Header ist ein Standard-COSE-Header und daher immer im geschützten Header-Map mit der Ganzzahl `1` als Label, wie in der IANA COSE Header Parameters Registry festgelegt. Die Literalzeichenfolge „`alg`" wird niemals als Label verwendet. Das Element „`sign_protected`" wird bei Verwendung von `COSE_Sign1` immer weggelassen.

Alle digitalen Signaturen in C2PA-Strukturen müssen eine `COSE_Sign1_Tagged`-Struktur gemäß der Definition in RFC 8152, Abschnitt 4.2. `COSE_Sign1_Tagged` enthält eine COSE_Sign1-Struktur. Für die Konstruktion von `COSE_Sign1_Tagged` gelten die folgenden zusätzlichen Anforderungen:

- Der gleiche `alg`-Header in der obigen `Sig_Struktur` muss im `geschützten` Header-Bucket vorhanden sein.
- Der Wert für das Payload-Feld und die Angabe, ob die Nutzlast in der Signatur vorhanden oder getrennt ist, werden bei jeder Verwendung digitaler Signaturen in dieser Spezifikation angegeben. Wenn die `Nutzlast` als getrennt angegeben ist, muss ihr Wert hier `nil` sein. Umgekehrt werden, wenn die Nutzlast in der Signatur vorhanden ist, die binären Inhalte der Nutzlast in diesem Feld als `bstr` gespeichert.

HINWEIS	COSE definiert `nil` in RFC 8152, Abschnitt 1.3, als Haupttyp 7, Wert 22 und verwendet diesen Wert. ausschließlich für getrennte Inhalte. Ein Byte-Array (Haupttyp 2) der Länge Null kann nicht zur Angabe getrennter Inhalte verwendet werden.

13.2.4. Hinzufügen einer angegebenen Signaturzeit

Ein Anspruchgenerator kann auch eine „behauptete Signaturzeit" festlegen, indem er einen iat-geschützten Header hinzufügt, dessen Wert ein `NumericDate` ist. Wenn vorhanden, muss dieser die Zeit darstellen, zu der die Signatur generiert wurde.

HINWEIS	Ein `NumericDate` ist ein numerisches CBOR-Datum (wie in RFC 8949, Abschnitt 3.4.2 beschrieben), jedoch ohne das führende Tag 1 (epochbasiertes Datum/Uhrzeit). Es wird an keiner anderen Stelle in dieser Spezifikation verwendet.
HINWEIS	Diese Empfehlung basiert auf laufenden Aktualisierungen von JAdES zur Bereitstellung eines nicht vertrauenswürdigen Zeitstempels, der nicht für die Überprüfung der Gültigkeit von Zertifikaten verwendet wird, aber in einer Benutzererfahrung verwendet werden könnte. Dies könnte in Szenarien nützlich sein, in denen der Anspruchgenerator keinen Zugriff auf eine vertrauenswürdige Zeitquelle hat, aber dennoch eine Zeit der Unterzeichnung angeben möchte.

13.2.5. Signaturvalidierung

Wenn bei der Erstellung einer Signatur der Anspruchgenerator auch als Validierer fungieren kann, sollte der Anspruchgenerator überprüfen, ob die Signaturberechtigung gemäß Kapitel 14, *Vertrauensmodell*, akzeptabel ist, und eine Warnung ausgeben, wenn dies nicht der Fall ist. Der Anspruchgenerator kann dennoch die Signatur mit dieser Berechtigung zulassen, wenn dies gewünscht wird. Dies kann wünschenswert sein, wenn bekannt ist, dass der Validierer des lokalen Anspruchgenerators eine andere Konfiguration hat als die Validierer, die von den erwarteten Nutzern des Assets verwendet werden.

13.2.6. Kryptografische Validierung

Bei der Überprüfung einer Signatur wird eine `Sig_Struktur` im Speicher generiert. Ihr Feld „`body_protected`“ wird mit dem Inhalt des `geschützten` Header-Buckets aus der Struktur `COSE_Sign1_Tagged` (RFC 8152, Abschnitt 4.4) gefüllt. Für das Feld „`payload`“ gilt: Wenn die Nutzlast in der Signatur als vorhanden angegeben wurde, wird sie aus dem Feld „`payload`“ der Struktur `COSE_Sign1_Tagged` gefüllt. Wenn die Nutzlast als getrennt angegeben wurde, ist das Feld „`payload`“ der Struktur `COSE_Sign1_Tagged leer.` In diesem Fall wird der Inhalt des Feldes „`payload`“ der `Sig_structure` aus derselben externen Quelle gefüllt, die bei der Generierung der Signatur verwendet wurde. Diese sind an den Stellen definiert, an denen die digitale Signatur in dieser Spezifikation verwendet wird.

13.2.7. Einfügen von Signatur-Symbolen

Ein C2PA-Manifest-Verbraucher möchte möglicherweise ein Symbol oder Logo für den Unterzeichner anzeigen. Um eine solche Grafik zu finden, muss er im eingebetteten Zertifikat nach einem Logotyp suchen, wie in RFC 9399 definiert. Ist kein Logotyp vorhanden, kann der Manifest-Verbraucher Symbole oder Logos aus anderen Quellen in einer implementierungsabhängigen Weise verwenden.

Kapitel 14. Vertrauensmodell

HINWEIS In diesem Abschnitt bezieht sich „Benutzer” auf menschliche Akteure, die C2PA-konforme Validatoren in Konsum- und Autorisierungsszenarien verwenden.

14.1. Übersicht

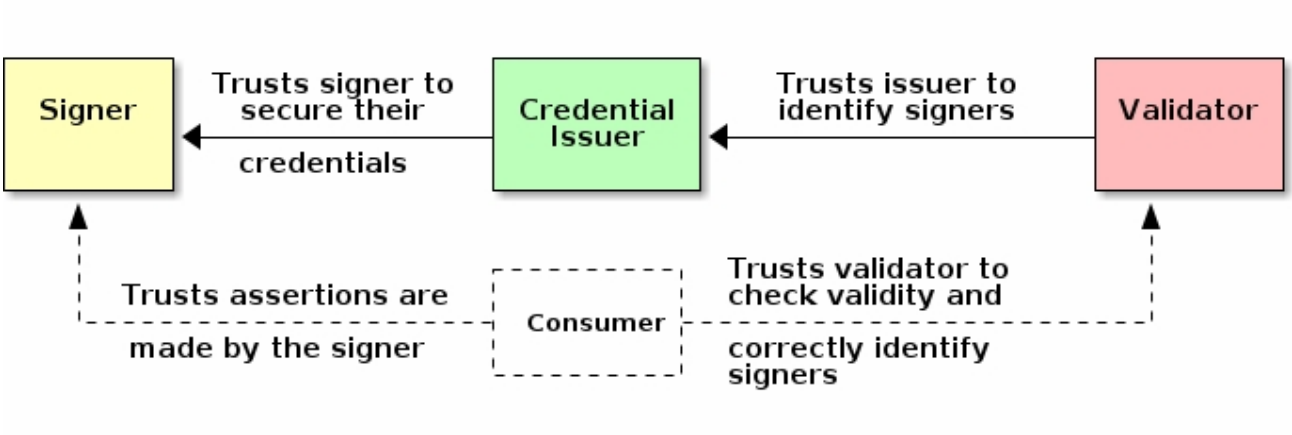

Abbildung 12. C2PA-Vertrauensmodell-Diagramm

Abbildung 12, „C2PA-Vertrauensmodell-Diagramm“, zeigt in Gelb, Grün und Rot die drei im Vertrauensmodell angegebenen Entitäten, die sich mit dem Vertrauen in die Identität eines Unterzeichners befassen. In gestrichelten Linien darunter befindet sich der Verbraucher (der im Vertrauensmodell nicht angegeben ist), der die Identität des Unterzeichners zusammen mit anderen Vertrauenssignalen verwendet, um zu entscheiden, ob die über einen Vermögenswert gemachten Aussagen wahr sind.

14.2. Identität der Unterzeichner

Die Identität im Vertrauensmodell ist das Mittel, mit dem ein kryptografischer Signaturschlüssel (auch als Berechtigungsnachweis bezeichnet) mit dem Unterzeichner verknüpft wird, um Vertrauensentscheidungen auf der Grundlage der Anspruchsignatur oder einer beliebigen Struktur (einschließlich, aber nicht beschränkt auf Behauptungen und Ansprüche) zu treffen, die mit diesem Schlüssel signiert wurde.

Die Berechtigungsnachweise müssen in den COSE-geschützten Headern der Struktur `COSE_Sign1_Tagged` aufgeführt sein, die für digitale Signaturen in allen C2PA-Manifesten verwendet wird. Genau eine Instanz eines Identitätsnachweises darf in der Vereinigung der geschützten und ungeschützten Header vorkommen. COSE_Sign1_Tagged-Strukturen ohne Berechtigungsnachweise oder mit zwei oder mehr Berechtigungsnachweisen sind abzulehnen. Die mehrfache Wiederholung desselben Berechtigungsnachweises, einschließlich separat in den geschützten und ungeschützten Headern, gilt ebenfalls als zwei oder mehr Berechtigungsnachweise und ist abzulehnen.

HINWEIS Ältere Versionen dieser Spezifikation erlaubten auch das Erscheinen der Berechtigung in den ungeschützten COSE-Headern.

Wie die Berechtigung im Header-Wert gespeichert wird, wie Vertrauenskette aufgebaut sind, ist in Abschnitt 14.5, „X.509-Zertifikate“, festgelegt.

14.3. Validierungszustände

14.3.1. Allgemeines

Ein Validator ist ein Manifest Consumer, der einige Validierungsaussagen über dieses Asset und das zugehörige aktive Manifest trifft. Der Prozess zum Abrufen dieser Aussagen wird im Abschnitt zur Validierung beschrieben. Der Akteur, der das Asset konsumiert, in der Regel über seinen User Agent und dessen Benutzeroberfläche, muss diese Aussagen dann interpretieren, um zu einer Reihe eigener Schlussfolgerungen über die Herkunft des von ihm konsumierten Assets zu gelangen. Diese Schlussfolgerungen werden aus diesen Aussagen und dem Inhalt des Assets selbst gezogen.

14.3.2. Manifest-Zustände

Basierend auf diesen Aussagen kann ein C2PA-Manifest eine der folgenden Formen haben:

- Wohlgeformt
- Gültig
- Vertrauenswürdig

HINWEIS Jedes vertrauenswürdige Manifest ist auch gültig, und jedes gültige Manifest ist auch wohlgeformt.

14.3.3. Asset-Zustände

Wenn ein Validierer meldet, dass die Teile des Assets, die durch Inhaltsbindungen abgedeckt sind, seit der Erstellung des aktiven Manifests [Abschnitt 15.12, „Validieren des Inhalts des Assets“] nicht geändert wurden und sein aktives Manifest entweder gültig oder vertrauenswürdig ist, dann ist das Asset selbst ein gültiges Asset.

14.3.4. Wohlgeformtes Manifest

Ein C2PA-Manifest ist wohlgeformt, wenn die Validierung ergibt, dass alle folgenden Punkte zutreffen:

- Der Inhalt des Manifests entspricht den normativen Anforderungen dieser Spezifikation, die durch den Validierungsprozess validiert werden.
- Es sind nur die für den spezifischen Typ des Manifests zulässigen Aussagen vorhanden [Abschnitt 15.10.1, „Validieren der korrekten Aussagen für den Manifesttyp“].
- Die Angaben im Manifest erfüllen alle Anforderungen an Angaben [Abschnitt 15.10.3, „Validierung von Angaben“].
- Alle im Manifest aufgeführten Inhaltsstoffe erfüllen alle Anforderungen an Inhaltsstoffe [Abschnitt 15.11, „Validierung der Inhaltsstoffe“].

14.3.5. Gültiges Manifest

Ein C2PA-Manifest ist gültig, wenn die Validierung ergibt, dass alle folgenden Punkte zutreffen:

- Das Manifest ist wohlgeformt [Abschnitt 14.3.4, „Wohlgeformtes Manifest“].
- Das Manifest wurde seit seiner Signierung nicht verändert [Abschnitt 13.2.6, „Kryptografische Validierung“].
- Die Anspruchsignatur erhält den Erfolgscode `claimSignature.validated` [Abschnitt 15.7, „Validieren der Signatur“].
- Validierung von des Anspruch Signatur Gültigkeit Zeitraum erhält die Erfolg Code von `claimSignature.insideValidity` [Abschnitt 15.8, „Zeitstempel validieren“].
- Die Berechtigung des Unterzeichners des C2PA-Manifests wird nicht mit einem Fehlercode von `signingCredential.ocsp.revoked` oder `signingCredential.ocsp.unknown` abgelehnt [Abschnitt 15.9, „Validieren der Informationen zur Berechtigungsentziehung“].

Wenn ein C2PA-Manifest gültig ist, kann die Angabe im Manifest dem Anspruchgenerator zugeordnet werden, der durch das Feld Feld `„claim_generator_info”` des Anspruchs identifiziert wird [Abschnitt 10.2.3, „Informationen zum Anspruchsgenerator”].

14.3.6. Vertrauenswürdiges Manifest

Ein C2PA-Manifest ist vertrauenswürdig, wenn die Validierung ergibt, dass alle folgenden Bedingungen erfüllt sind:

- Das Manifest ist gültig [Abschnitt 14.3.5, „Gültiges Manifest“].
- Die Signaturberechtigung des C2PA-Manifests erhält den Erfolgscode `„signingCredential.trusted“.` [Abschnitt 15.7, „Signatur validieren“].

14.4. Vertrauenslisten

14.4.1. C2PA-Unterzeichner

Ein Validierer muss die folgenden Informationen zur Bewertung von C2PA-Signaturnutzern pflegen:

- Eine Liste der akzeptierten Werte für die erweiterte Schlüsselverwendung (Extended Key Usage, EKU).
- Für jeden akzeptierten EKU-Wert eine Liste der X.509-Zertifikatsvertrauensanker.

Für den EKU `c2pa-kp-claimSigning` (1.3.6.1.4.1.62558.2.1) muss die Liste der Vertrauensanker unter anderem die von C2PA bereitgestellten Vertrauensanker für Unterzeichner enthalten (d. h. die C2PA-Vertrauensliste).

HINWEIS Einige dieser Listen können leer sein.

Zusätzlich zu der Liste der Vertrauensanker für die EKU `c2pa-kp-claimSigning`, die in der C2PA-Vertrauensliste bereitgestellt wird, sollte ein Validator einem Benutzer die Konfiguration zusätzlicher Vertrauensanker für diese EKU und/oder für andere EKUs (z. B. `id-kp-emailProtection` (1.3.6.1.5.5.7.3.4) oder `id-kp-documentSigning` (1.3.6.1.5.5.7.3.36)). Ein Validator sollte Standardoptionen bereitstellen oder von externen Parteien gepflegte Listen anbieten, die der Benutzer auswählen kann, um den Trust-Anchor-Speicher des Validators für C2PA-Signierende zu füllen.

HINWEIS Frühere Versionen dieser Spezifikation erforderten das Vorhandensein von `id-kp-emailProtection` oder `id-`

`kp-documentSigning` erforderlich. Durch die Aufnahme mindestens einer dieser beiden EKUs in das Zertifikat eines Unterzeichners zusammen mit `c2pa-kp-claimSigning` kann die Kompatibilität mit älteren Validatoren verbessert werden.

14.4.2. Zeitstempelbehörden

Ein Validierer muss eine Liste von X.509-Zertifikatsvertrauensankern für Zeitstempelbehörden (TSAs) führen, die von den Listen für C2PA-Signatoren getrennt sein muss. Diese Liste muss unter anderem die von der C2PA bereitgestellten Zeitstempelbehörden-Vertrauensanker enthalten (d. h. die C2PA TSA-Vertrauensliste).

HINWEIS Diese Liste kann leer sein.

Zusätzlich zu der Liste der Vertrauensanker, die in der C2PA TSA-Vertrauensliste aufgeführt sind, sollte ein Validierer einem Benutzer die Konfiguration zusätzlicher TSA-Vertrauensankerspeicher ermöglichen und Standardoptionen oder von externen Parteien gepflegte Listen bereitstellen, die der Benutzer auswählen kann, um den Vertrauensankerspeicher des Validierers für Zeitstempelbehörden zu füllen.

14.4.3. Private Speicherung von Anmeldedaten

Ein Validator kann dem Benutzer auch erlauben, einen privaten Speicher für Signatur-Anmeldedaten anzulegen und zu verwalten. Dieser Speicher ist als „Adressbuch" für Anmeldedaten gedacht, denen der Benutzer aufgrund einer Out-of-Band-Beziehung vertraut. Falls vorhanden, gilt der private Berechtigungsnachweis-Speicher nur für die Validierung signierter C2PA-Manifeste und nicht für die Validierung von Zeitstempeln. Falls vorhanden, erlaubt der private Berechtigungsnachweis-Speicher nur das direkte Vertrauen in Signaturberechtigungsnachweise; Einträge im privaten Berechtigungsnachweis-Speicher können keine Berechtigungsnachweise ausstellen und dürfen während der Validierung nicht als Vertrauensanker verwendet werden.

Ein Validator darf nicht mit Einträgen in einem privaten Credential Store vorkonfiguriert sein.

Ein Validator darf nur als Reaktion auf eine Benutzeranforderung, der Berechtigung zu vertrauen, Einträge zu einem privaten Berechtigungsnachweis-Speicher hinzufügen. Ebenso darf ein Validator nur als Reaktion auf eine Benutzeranforderung, der Berechtigung nicht mehr zu vertrauen, Einträge aus einem privaten Berechtigungsnachweis-Speicher entfernen.

14.5. X.509-Zertifikate

X.509-Zertifikate werden gemäß der Definition in RFC 9360 (*CBOR Object Signing and Encryption (COSE): Header Parameters for Carrying and Referencing X.509 Certificates*) gespeichert. Der Einfachheit halber wird die Definition von `x5chain` unten wiedergegeben.

WICHTIG Diese Spezifikation enthält zusätzliche Anforderungen, die über die Anforderungen von RFC 9360 hinausgehen und nach dem zitierten Text aufgeführt sind. Insbesondere verlangt diese Spezifikation, dass alle Zwischenzertifikate in den `x5`chain-Header aufgenommen werden müssen und verlangt, dass Anspruchgeneratoren den x5chain-Header immer im geschützten Header-Bucket platzieren.

> x5chain: Dieser Header-Parameter enthält ein geordnetes Array von X.509-Zertifikaten. Die Zertifikate sind in der Reihenfolge anzuordnen, beginnend mit dem Zertifikat, das den Endentitäts-Schlüssel enthält, gefolgt von dem Zertifikat, das es signiert hat, und so weiter. Es besteht keine Anforderung für das gesamte

> Kette im Element vorhanden sein, wenn Grund zu der Annahme besteht, dass die vertrauende Partei bereits über die fehlenden Zertifikate verfügt oder diese finden kann. Das bedeutet, dass die vertrauende Partei weiterhin den Pfad aufbauen muss, aber dass in diesem Header-Parameter ein Kandidat für den Pfad vorgeschlagen wird.
>
> Der Vertrauensmechanismus MUSS alle Zertifikate in diesem Parameter als nicht vertrauenswürdige Eingabe verarbeiten. Das Vorhandensein eines selbstsignierten Zertifikats im Parameter DARF NICHT ohne eine Bestätigung außerhalb des Bandes zur Aktualisierung der Vertrauensanker führen. Da der Inhalt dieses Header-Parameters eine nicht vertrauenswürdige Eingabe ist, kann sich der Header-Parameter entweder im geschützten oder im ungeschützten Header-Bucket befinden. Das Senden des Header-Parameters im ungeschützten Header-Bucket ermöglicht es einem Vermittler, Zertifikate zu entfernen oder hinzuzufügen.
>
> Das Endentitätszertifikat MUSS durch COSE integritätsgeschützt sein. Dies kann beispielsweise durch Senden des Header-Parameters im geschützten Header, durch Senden eines „x5chain" im ungeschützten Header in Kombination mit einem „x5t" im geschützten Header oder durch Einfügen des Endentitätszertifikats in die external_aad erfolgen.
>
> Dieser Header-Parameter ermöglicht es, ein einzelnes X.509-Zertifikat oder eine Kette von X.509-Zertifikaten in der Nachricht zu übertragen.
>
> - Wenn ein einzelnes Zertifikat übertragen wird, wird es in einer CBOR-Byte-Zeichenkette platziert.
> - Wenn mehrere Zertifikate übertragen werden, wird ein CBOR-Array von Byte-Zeichenfolgen verwendet, wobei jedes Zertifikat in einer eigenen Byte-Zeichenfolge enthalten ist.

s wird erwartet, dass der Validator nur über die Zertifikate für seine Vertrauensanker verfügt. Daher muss der Anspruchgenerator beim rstellen des x5chain-Headers als Teil der Signatur das Zertifikat des Unterzeichners und alle Zwischenzertifizierungsstellen in den Wert des eaders aufnehmen. Das Zertifikat des Vertrauensankers (auch als Stammzertifikat bezeichnet) sollte nicht enthalten sein.

Das Element „`subjectPublicKeyInfo`" des ersten oder einzigen Zertifikats ist der öffentliche Schlüssel, der zur Validierung der Signatur verwendet wird. Das Element „`validity`" der Sequenz „`tbsCertificate`" gibt die Gültigkeitsdauer des Zertifikats an.

Eine frühere Version dieser Spezifikation verlangte von Anspruchgeneratoren, nur die Zeichenfolge „`x5chain`" zu schreiben, um die unwahrscheinliche Möglichkeit zu vermeiden, dass die Ganzzahl „`33`" nicht standardisiert würde.

Die Ganzzahl-Kennzeichnung `33` wurde nun standardisiert, und diese Spezifikation übernimmt sie nun als Standard und verwirft die Verwendung der Zeichenfolgenkennzeichnung. Daher gilt:

- Claim-Generatoren sollten nur die Ganzzahl `33` als Label verwenden, wenn sie diesen Header in eine COSE-Signatur einfügen. Claim-Generatoren können weiterhin das String-Label `x5chain` schreiben, aber dieses Verhalten ist nun veraltet und Claim-Generatoren

Generatoren sollten so aktualisiert werden, dass sie nur noch die Ganzzahl-Kennung verwenden. Anspruchgeneratoren müssen diesen Header gemäß den oben genannten Anforderungen nur im geschützten Header-Bucket der COSE-Signatur platzieren.

- Validatoren müssen entweder die Zeichenfolge „x5chain“ oder die ganze Zahl „33“ als Bezeichnung für diesen Header akzeptieren. Sind beide Bezeichnungen vorhanden, müssen Validatoren den Header mit der ganzen Zahl „33“ als Bezeichnung verwenden und den Header mit der Zeichenfolge „x5chain“ als Bezeichnung ignorieren. Validatoren müssen den Header entweder aus dem geschützten oder dem ungeschützten Bucket akzeptieren, um die Kompatibilität mit früheren Versionen dieser Spezifikation zu gewährleisten. In Übereinstimmung mit Abschnitt 14.2, „Identität der Unterzeichner“, muss ein Validator die Anspruchsignatur als fehlerhaft ablehnen, wenn dieser Header sowohl im geschützten als auch im ungeschützten Bucket mit derselben Bezeichnung erscheint, da mehrere Anmeldedaten vorhanden sind.

14.5.1. Zertifikatsprofile

14.5.1.1. Allgemeine Anforderungen

In diesem Abschnitt werden die Anforderungen definiert, anhand derer überprüft wird, ob ein X.509-Zertifikat als Signaturzertifikat gemäß Abschnitt 15.7, „Überprüfen der Signatur“, akzeptiert werden kann.

Alle Zertifikate müssen die folgenden Anforderungen erfüllen.

- Das Feld „algorithm“ des Feldes „signatureAlgorithm“ muss einen der folgenden Werte enthalten:

ecdsa-with-SHA256

RFC 5758, Abschnitt 3.2

ecdsa-with-SHA384

RFC 5758, Abschnitt 3.2

ecdsa-with-SHA512

RFC 5758, Abschnitt 3.2

sha256WithRSAEncryption

RFC 8017, Anhang A.2.4

sha384WithRSAEncryption

RFC 8017, Anhang A.2.4

sha512WithRSAEncryption

RFC 8017, Anhang A.2.4

id-RSASSA-PSS

RFC 8017, Anhang A.2.3

id-Ed25519

RFC 8410 Abschnitt 3

- Wenn das Feld „algorithm“ des Feldes „signatureAlgorithm“ den Wert „id-RSASSA-PSS“ hat, ist das Feld „parameters“ vom Typ „RSASSA-PSS-params“. Seine Felder müssen die folgenden Anforderungen gemäß RFC 8017, Anhang A.2.3 erfüllen:
  - Das Feld „hashAlgorithm“ muss vorhanden sein.
  - Das Feld „algorithm“ des Feldes „hashAlgorithm“ muss einen der folgenden Werte haben, wie in RFC 8017, Anhang B.1 definiert:
    - id-sha256.
    - id-sha384.
    - id-sha512.
  - Das Feld „maskGenAlgorithm“ muss vorhanden sein.
  - Das Feld „algorithm“ des Feldes „parameters“ des Feldes „maskGenAlgorithm“ muss mit dem Algorithmusfeld des hashAlgorithm-Feldes übereinstimmen.
- Wenn das Feld „Algorithm“ des Feldes „subjectPublicKeyInfo“ des Zertifikats den Wert „id-ecPublicKey“ hat, muss das Feld „Parameters“ eine der folgenden benannten Kurven aus RFC 5480, Abschnitt 2.1.1.1 enthalten:
  - prime256v1.
  - secp384r1.
  - secp521r1.
- Wenn das Algorithmusfeld des Zertifikats subjectPublicKeyInfo rsaEncryption oder id-RSASSA-PSS lautet, muss das Modulfeld des Parameterfelds eine Länge von mindestens 2048 Bit haben.

Alle Zertifikate mit Ausnahme derjenigen im privaten Zertifikatsspeicher für X.509-Zertifikate müssen die folgenden zusätzlichen Anforderungen erfüllen, um akzeptiert zu werden.

- Die Version muss v3 gemäß RFC 5280, Abschnitt 4.1.2.1, sein.
- Die optionalen Felder issuerUniqueID und subjectUniqueID der TBSCertificate-Sequenz dürfen gemäß RFC 5280, Abschnitt 4.1.2.8, nicht vorhanden sein.
- Die Erweiterung „Basic Constraints“ muss RFC 5280, Abschnitt 4.2.1.9 entsprechen. Insbesondere muss eine der folgenden Bedingungen erfüllt sein:
  - Wenn der zertifizierte öffentliche Schlüssel zur Überprüfung von Zertifikatssignaturen verwendet werden kann, muss die Erweiterung „Basic Constraints“ mit dem booleschen Wert „cA“ vorhanden sein.
  - Wenn der zertifizierte öffentliche Schlüssel nicht zur Überprüfung von Zertifikatssignaturen verwendet werden kann, darf die Erweiterung „Basic Constraints“ entweder nicht vorhanden sein oder der boolesche Wert „cA“ in der Erweiterung darf nicht gesetzt sein und das Bit „keyCertSign“ in der Erweiterung „key usage“ darf nicht gesetzt sein.
- Die Erweiterung „Authority Key Identifier“ muss gemäß RFC 5280, Abschnitt 4.2.1.1, in jedem Zertifikat vorhanden sein, das nicht selbstsigniert ist.

- Gemäß RFC 5280, Abschnitt 4.2.1.2, muss die Erweiterung „Subject Key Identifier“ in jedem Zertifikat vorhanden sein, das als CA fungiert. Sie sollte in Endentitätszertifikaten vorhanden sein.
- Gemäß RFC 5280, Abschnitt 4.2.1.3, muss die Erweiterung „Key Usage“ vorhanden sein und als kritisch gekennzeichnet werden. Zertifikate, die zum Signieren von C2PA-Manifesten verwendet werden, müssen das Bit `„digitalSignature“` setzen. Das Bit `„keyCertSign“` darf nur gesetzt werden, wenn das Bit `„cA` boolean“ in der Erweiterung „Basic Constraints“ gesetzt ist.
- Die Erweiterung „Extended Key Usage“ (EKU) muss in jedem Zertifikat vorhanden und nicht leer sein, in dem die Erweiterung „Basic Constraints“ fehlt oder der cA-Boolesche Wert nicht gesetzt ist, gemäß RFC 5280, Abschnitt 4.2.1.12. Diese werden üblicherweise als „Endentity“- oder „Leaf“-Zertifikate bezeichnet.
    - Die EKU `„anyExtendedKeyUsage“` (2.5.29.37.0) darf nicht vorhanden sein.
    - Ein Zertifikat, das Gegenzeichnungen für Zeitstempel signiert, muss für den Zweck `„id-kp-timeStamping` (1.3.6.1.5.5.7.3.8) gültig sein.
    - Ein Zertifikat, das OCSP-Antworten für Zertifikate signiert, muss für den Zweck `id-kp-OCSPSigning` (1.3.6.1.5.5.7.3.9) gültig sein.
    - Wenn ein Zertifikat entweder für `id-kp-timeStamping` oder `id-kp-OCSPSigning` gültig ist, muss es für genau einen dieser beiden Zwecke gültig sein und darf für keinen anderen Zweck gültig sein.
    - Ein Zertifikat sollte für keine anderen als die oben genannten Zwecke gültig sein, aber das Vorhandensein von EKUs, die nicht in diesem Profil und nicht in der Liste der EKUs im Konfigurationsspeicher aufgeführt sind, darf nicht dazu führen, dass das Zertifikat abgelehnt wird.

14.5.1.2. Zertifikat-Vertrauenskette

Bei der Validierung eines Zertifikats als Signatur-Anmeldeinformation wird das Zertifikat akzeptiert, wenn es im privaten Anmeldeinformationsspeicher für
X.509-Zertifikate vorhanden ist, wird das Zertifikat akzeptiert. Der private Berechtigungsnachweisspeicher wird bei der Validierung von Zeitstempeln nicht konsultiert.

Wenn das Zertifikat nicht im privaten Zertifikatsspeicher vorhanden ist oder der Validierer keinen solchen implementiert, muss die Vertrauenskette gemäß dem Verfahren in RFC 5280, Abschnitt 6 für den jeweiligen Zweck (Signierung, Zeitstempelung oder OCSP-Signierung) und für den für diesen Zweck geeigneten Vertrauensanker-Speicher aufgebaut und validiert werden. Jeder Fehler dieses Validierungsalgorithmus bedeutet, dass die Kette abgelehnt wird. Der private Zertifikatspeicher wird beim Aufbau von Zertifikatsketten niemals berücksichtigt; Zertifikate im privaten Zertifikatspeicher können nicht als Zertifizierungsstellen fungieren.

Zur Signierung von C2PA-Ansprüchen oder Zeitstempeln dürfen nur Endentitätszertifikate verwendet werden. Ein CA-Zertifikat darf für diese Zwecke nicht verwendet werden. Jedes CA-Zertifikat (bei dem der boolesche Wert `„cA“` in der Erweiterung „Basic Constraints“ gesetzt ist), das zur Validierung einer Signatur auf einem C2PA-Anspruch, einem Zeitstempel oder einer OCSP-Antwort verwendet wird, muss mit dem Fehlercode `„signingCredential.untrusted“` abgelehnt werden.

Ein Validierer muss sicherstellen, dass ein Signaturzertifikat für den Zweck, für den es verwendet wird, autorisiert ist, und Zertifikate ablehnen, die für einen nicht autorisierten Zweck verwendet werden. Ein Zertifikat ist für einen bestimmten Zweck autorisiert, wenn der EKU-Objektbezeichner (OID) dieses Zwecks in der Erweiterung „Extended Key Usage“ des Zertifikats enthalten ist (RFC 5280, Abschnitt 4.2.1.12).

Bei der Validierung eines Zertifikats, das zur Signierung eines C2PA-Anspruchs verwendet wird, muss das Signaturzertifikat mindestens eine der EKUs aufweisen, für die der Validierer über eine zugehörige Liste von Vertrauensankern verfügt (siehe Abschnitt 14.4.1, „C2PA-Signatare“), und der Validierer darf

nur die Vertrauensanker verwenden, die er mit den im Zertifikat vorhandenen EKUs verknüpft.

Bei der Validierung einer Zertifikatskette, die zum Signieren eines Zeitstempels verwendet wird, muss das Signaturzertifikat über die EKU `id-kp-timeStamping` (1.3.6.1.5.5.7.3.8) verfügen.

Bei der Validierung einer Zertifikatskette, die zum Signieren einer OCSP-Antwort verwendet wird, muss das Signaturzertifikat die EKU `id-kp-OCSPSigning` (1.3.6.1.5.5.7.3.9) aufweisen.

Mit Ausnahme von Zertifikaten, die über den privaten Zertifikatsspeicher für X.509-Zertifikate akzeptiert werden, muss ein Validierer die Konformität eines Zertifikats mit dem Zertifikatsprofil überprüfen und nicht konforme Zertifikate ablehnen. Dazu gehört, dass die Erweiterung „Extended Key Usage“ vorhanden sein muss und dass ein Zertifikat für höchstens einen der drei in diesem Abschnitt aufgeführten Zwecke autorisiert sein darf: C2PA-Signierung, Zeitstempel-Signierung oder OCSP-Antwort-Signierung.

Wie im Zertifikatsprofil beschrieben, müssen Zertifikate von Zertifizierungsstellen (CA), die Zertifikate ausstellen, keine EKU-Erweiterung haben und haben diese in der Regel auch nicht. Ist eine solche vorhanden, wird sie ignoriert. Diese Anforderung gilt nur für Endentitätszertifikate, die C2PA-Manifeste, Zeitstempel oder OCSP-Antworten signieren. CA-Zertifikate dürfen nicht zum Signieren von C2PA-Manifesten, Zeitstempeln oder OCSP-Antworten verwendet werden.

14.5.2. Zertifikatswiderruf

X.509-Zertifikate unterstützen Abfragen zum Widerrufsstatus. Ein Anspruchgenerator sollte das Online Certificate Status Protocol (OCSP, RFC 6960) und OCSP-Stapling (wie ursprünglich in RFC 6066, Abschnitt 8 konzipiert, aber wie in dieser Klausel beschrieben implementiert) verwenden, um den Widerruf zu implementieren. Der Anspruchgenerator darf keine Zertifikatssperrlisten (CRLs, RFC 5280) verwenden. ``

HINWEIS

Die Verwendung von CRLs erfordert das Herunterladen der gesamten Liste der widerrufenen Zertifikate für jede Zertifizierungsstelle.

, was sehr zeitaufwendig sein kann. Eine CRL könnte zwar auf die gleiche Weise wie eine OCSP-Antwort angehängt werden, doch aufgrund der potenziellen Größe einer CRL im Vergleich zu einer OCSP-Antwort ist dies ebenfalls nicht wünschenswert.

Eine konforme Zertifizierungsstelle sollte eine AuthorityInfoAccess (AIA)-Erweiterung (RFC 5280, Abschnitt 4.2.2.1) in ihre ausgestellten Zertifikate aufnehmen, um Zugriffsinformationen für den von der Zertifizierungsstelle betriebenen OCSP-Dienst bereitzustellen.

Wenn das Zertifikat eine AIA-Erweiterung enthält, müssen die Sperrinformationen in einem ungeschützten Header der COSE_Sign1-Struktur mit der Zeichenfolgenbezeichnung `rVals` gespeichert werden, und das Schema des Werts muss der rVals-Regel in Beispiel 3, „CDDL für rVals”, entsprechen:

Beispiel 3. CDDL für `rVals`

```
; CBOR-Version von rVals und zugehörigen Strukturen basierend auf dem JSON-Schema in
https://www.etsi.org/deliver/etsi_ts/119100_119199/11918201/01.01.01_60/ts_11918201v010 101p.pdf Abschnitt
5.3.5.2
rVals = {
  „ocspVals”: [1* bstr]
}
```

HINWEIS	Die obige Definition ist eine CBOR-Anpassung einer Teilmenge des Schemas aus JAdES, Abschnitt 5.3.5.2, das nur OCSP-Antworten speichert und diese als Binärzeichenfolgen speichert.

Bevor ein Anspruch signiert wird, sollte ein Anspruchgenerator, wenn das Zertifikat eines Unterzeichners die AIA-Erweiterung enthält, den darin angegebenen OCSP-Dienst abfragen, die Antwort erfassen und sie in einem Element des ocspVals-Arrays des rVals-Headers speichern. Der Anspruchgenerator sollte dasselbe für alle Zwischen-CA-Zertifikate tun, die er in die Anspruchsignatur einbezieht.

Nach Erhalt des Anspruchs werden die angehängten OCSP-Antworten gemäß Abschnitt 3.2 von RFC 6960 validiert.

Der Prozess zur Validierung des Widerrufsstatus eines Zertifikats nach der Signierung eines Anspruchs wird unter „Validieren der Informationen zum Widerruf von Berechtigungsnachweisen“ näher beschrieben.

Kapitel 15. Validierung

15.1. Validierungsprozess

15.1.1. Beschreibung

Die Validierung eines C2PA-Manifests ist ein mehrstufiger Prozess, der die Validierung der darin enthaltenen Aussagen, Ansprüche und zugehörigen Anspruchsignaturen sowie (nur bei aktiven Manifesten) die Validierung aller zugehörigen Hard Bindings umfasst. Dieser Validierungsprozess wird von einem Validator durchgeführt, einem Hardware- oder Software-Akteur, der die in diesem Abschnitt beschriebenen Validierungsalgorithmen implementiert.

15.1.2. Phasen der Validierung

Diese Phasen, die in keiner bestimmten Reihenfolge aufgeführt sind, werden in den folgenden Abschnitten beschrieben:

- Abschnitt 15.10, „Validierung der Aussagen": Validierung der Aussagen.
- Abschnitt 15.11, „Validierung der Inhaltsstoffe": Validierung der Inhaltsstoffe, falls vorhanden.
- Abschnitt 15.8, „Zeitstempel validieren": Validierung des Zeitstempels.
- Abschnitt 15.9, „Validierung der Informationen zur Sperrung von Berechtigungsnachweisen": Validierung der Informationen zur Sperrung von Berechtigungsnachweisen.
- Abschnitt 15.7, „Signatur validieren": Validierung der Signatur der Behauptung.
- Abschnitt 15.12, „Inhalt des Assets validieren": Validierung des Inhalts des Assets.

Wie in Abschnitt 14.3, „Validierungsstatus", beschrieben, kann ein C2PA-Manifest auf der Grundlage der Ergebnisse dieser Schritte als wohlgeformt, gültig oder vertrauenswürdig eingestuft werden.

15.1.3. Visuelle Darstellung

Abbildung 13, „Validierung einer Forderung", ist eine visuelle Darstellung des Prozesses zur Validierung eines C2PA-Manifests.

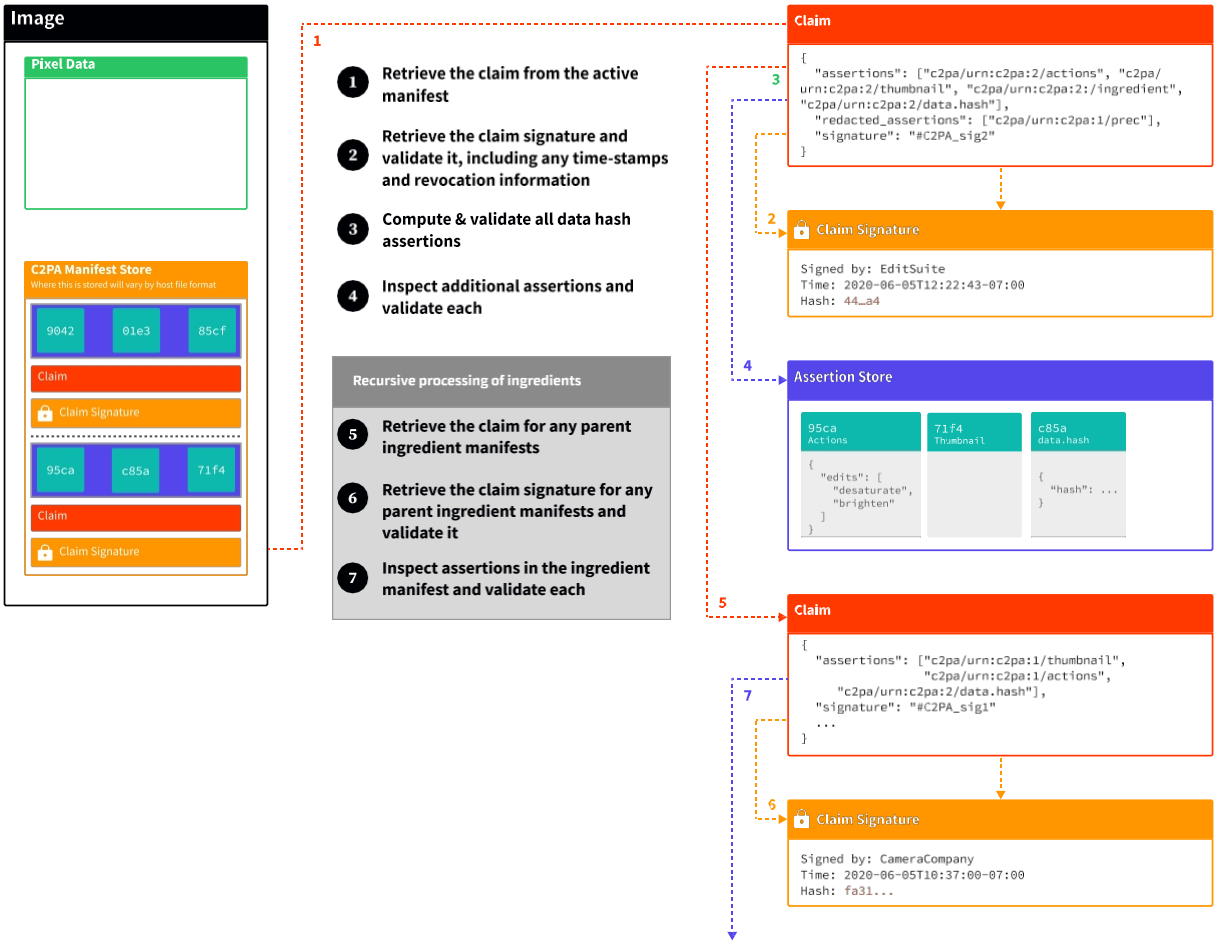

Abbildung 13. Validierung einer Forderung

HINWEIS

Bei Abweichungen zwischen der visuellen Darstellung und dem Text ist der Text maßgebend.

15.2. Rückgabe der Validierungsergebnisse

15.2.1. Allgemeines

Der Validierungsalgorithmus gibt einen konsolidierten Satz von Validierungsergebnissen für alle Manifeste im C2PA-Manifest-Speicher des Assets zurück, einschließlich des aktiven Manifests sowie aller anderen Manifeste im C2PA-Manifest-Speicher, auf die über Inhaltsangaben verwiesen wird.

Validierungsergebnisse werden über einen Standardsatz von Erfolgs-, Informations- und Fehlercodes ausgedrückt, wie unten in Abschnitt 15.2.2, „Standard-Statuscodes“, definiert. Benutzerdefinierte Statuscodes sind ebenfalls zulässig, wenn ein Anspruchgenerator bestimmte prozessspezifische Statusinformationen aufzeichnen muss. Benutzerdefinierte Codes müssen derselben Syntax entsprechen wie entitätsspezifische Namespaces, z. B. `com.litware`.

Wenn ein Anspruchgenerator einen Inhaltsstoff über eine Inhaltsstoffangabe hinzufügt, muss er als Validierer fungieren und den in diesem Abschnitt beschriebenen vollständigen Validierungsalgorithmus für den Inhaltsstoff durchführen. Der Anspruchgenerator muss die

Validierungs sergebnisse von der Zutat, gemäß der folgenden CDDL Definition Schema, als der Wert von der Feld „validationResults” in der Zutatenangabe.

CDDL für Validierungsergebnisse

```
; Validierungscodes

; Erfolgscodes
$status-code /= "assertion.accessible"
$status-code /= „assertion.bmffHash.match”
$status-code /= "assertion.boxesHash.match"
$status-code /= „assertion.collectionHash.match”
$status-code /= „assertion.dataHash.match”
$status-code /= „assertion.hashedURI.match”
$status-code /= „claimSignature.insideValidity”
$status-code /= „claimSignature.validated”
$status-code /= „Zutat.claimSignature.validated”
$status-code /= „Zutat.Manifest.validiert”
$status-code /= „signingCredential.ocsp.notRevoked”
$status-code /= „signingCredential.trusted”
$status-code /= „timeStamp.trusted”
$status-code /= „timeStamp.validated”

; Informationscodes
$status-code /= „Zutat.unbekannteHerkunft”
$status-code /= „manifest.unknownProvenance”
$status-code /= „signingCredential.ocsp.inaccessible”
$status-code /= „signingCredential.ocsp.skipped”
$status-code /= „timeOfSigning.insideValidity”
$status-code /= „timeOfSigning.outsideValidity”
$status-code /= „timeStamp.malformed”
$status-code /= „timeStamp.mismatch”
$status-code /= „timeStamp.outsideValidity”
$status-code /= „timeStamp.untrusted”
$status-code /= „assertion.dataHash.additionalExclusionsPresent”

; Fehlercodes
$status-code /= „algorithm.deprecated”
$status-code /= "algorithm.unsupported"
$status-code /= „assertion.action.ingredientMismatch”
$status-code /= „assertion.action.malformed”
$status-code /= „assertion.action.redacted”
$status-code /= „assertion.action.redactionMismatch”
$status-code /= „assertion.bmffHash.malformed”
$status-code /= „assertion.bmffHash.mismatch”
$status-code /= „assertion.boxesHash.mismatch”
$status-code /= „assertion.boxesHash.malformed”
$status-code /= „assertion.boxesHash.unknownBox”
$status-code /= „assertion.cloud-data.hardBinding”
$status-code /= „assertion.cloud-data.actions”
$status-code /= „assertion.cloud-data.hardBinding”
$status-code /= „assertion.cloud-data.malformed”
$status-code /= „assertion.collectionHash.incorrectFileCount”
$status-code /= „assertion.collectionHash.invalidURI”
$status-code /= „assertion.collectionHash.malformed”
$status-code /= „assertion.collectionHash.mismatch”
$status-code /= „assertion.dataHash.malformed”
$status-code /= „assertion.dataHash.mismatch”
$status-code /= „assertion.hashedURI.mismatch”
$status-code /= „assertion.inaccessible”
$status-code /= „assertion.ingredient.malformed”
```

```
$status-code /= „assertion.json.invalid”
$status-code /= „assertion.missing”
$status-code /= "assertion.multipleHardBindings"
$status-code /= „assertion.notRedacted”
$status-code /= „assertion.outsideManifest”
$status-code /= „assertion.selfRedacted”
$status-code /= „assertion.undeclared”
$status-code /= „claim.cbor.invalid”
$status-code /= „claim.hardBindings.missing”
$status-code /= „claim.malformed”
$status-code /= „claim.missing”
$status-code /= „claim.multiple”
$status-code /= „claimSignature.missing”
$status-code /= „claimSignature.mismatch”
$status-code /= „claimSignature.outsideValidity”
$status-code /= „general.error“ ; wenn nichts anderes zutrifft
$status-code /= „hashedURI.missing”
$status-code /= „hashedURI.mismatch”
$status-code /= „Zutat.claimSignature.fehlt”
$status-code /= „ingredient.claimSignature.mismatch”
$status-code /= „Zutat.Manifest fehlt”
$status-code /= „Zutat.Manifest.Nichtübereinstimmung”
$status-code /= „manifest.compressed.invalid”
$status-code /= „manifest.inaccessible”
$status-code /= „manifest.multipleParents”
$status-code /= „manifest.timestamp.invalid”
$status-code /= "manifest.timestamp.wrongParents"
$status-code /= „manifest.update.invalid”
$status-code /= „manifest.update.wrongParents”
$status-code /= „signingCredential.invalid”
$status-code /= „signingCredential.ocsp.revoked”
$status-code /= „signingCredential.ocsp.unknown”
$status-code /= „signingCredential.untrusted”

; benutzerdefinierte Statuscodes
$status-code /= tstr .regexp "([\\da-zA-Z_-]+\\.)+[\\da-zA-Z_-]+"

status-map = {
   "code": $status-code,               ; Eine beschriftete Zeichenfolge, die den Status beschreibt
   ? "url": jumbf-uri-type,           ; JUMBF-URI-Verweis auf die JUMBF-Box, für die dieser Statuscode gilt
   ? „explanation”: tstr .size (1..max-tstr-length), ; Eine für Menschen lesbare Zeichenfolge, die den Status
erklärt
   ? „success“: bool                   ; VERALTET. Gibt der Code Erfolg (true) oder Fehler (false)
zurück?
}

status-codes-map = {
     „success”: [* $status-map],                ; Ein Array mit Validierungserfolgscodes. Kann leer
sein.
     „informational”: [* $status-map],          ; ein Array mit Validierungsinformationscodes. Kann leer
sein.
     „failure”: [* $status-map]                 ; ein Array mit Validierungsfehlercodes. Kann leer
sein.
}

; Objekte, die Validierungsergebnisse für ein Manifest und dessen Inhaltsstoffe

enthalten validation-results-map = {
     ? „activeManifest”: $status-codes-map, ; Validierungsstatuscodes für das aktive Manifest der Komponente.
Vorhanden, wenn die Komponente ein C2PA-Asset ist. Nicht vorhanden, wenn die Komponente kein C2PA-Asset ist.
     ? „ingredientDeltas”: [* $ingredient-delta-validation-result-map] ; Liste aller
```

```
Änderungen/Deltas zwischen den aktuellen und früheren Validierungsergebnissen für das Manifest jedes
Inhaltsstoffs. Vorhanden, wenn der Inhaltsstoff ein C2PA-Asset ist.
}

ingredient-delta-validation-result-map = {
     „ingredientAssertionURI": jumbf-uri-type, ; JUMBF-URI-Verweis auf die Inhaltsstoffangabe
     „validationDeltas": $status-codes-map               ; Validierungsergebnisse für das aktive Manifest
der Zutat
}
```

15.2.2. Standard-Statuscodes

15.2.2.1. Erfolgscodes

Tabelle 2. Validierungserfolgscodes

Wert	Bedeutung	`url` Verwendung
`assertion.accessible`	Eine nicht eingebettete (remote) Assertion war zum Zeitpunkt der Validierung zugänglich.	C2PA-Assertion
`assertion.bmffHash.match`	Der Hash eines boxbasierten Assets stimmt mit dem in der BMFF-Hash-Assertion deklarierten Hash überein.	C2PA-Assertion
`assertion.boxesHash.match`	Der Hash eines boxbasierten Assets stimmt mit dem in der allgemeinen Box-Hash-Assertion deklarierten Hash überein.	C2PA-Assertion
`assertion.collectionHash.matc h`	Die Hashes aller in einer Sammlung enthaltenen Assets stimmen mit den in der Collection-Daten-Hash-Assertion deklarierten Hashes überein.	C2PA-Assertion
`assertion.dataHash.match`	Der Hash eines Bytebereichs des Assets stimmt mit dem in der Daten-Hash-Assertion deklarierten Hash überein.	C2PA-Assertion
`assertion.hashedURI.match`	Der Hash der referenzierten Assertion stimmt mit dem entsprechenden Hash in der gehashten URI der Assertion im Anspruch überein.	C2PA-Assertion
`assertion.multiAssetHash.matc h`	Der Hash eines Teils einer Multi-Asset-Hash-Assertion stimmt mit dem entsprechenden Hash in der Multi-Asset-Hash-Map der Assertion überein.	C2PA-Assertion
`claimSignature.insideValidity`	Die in der Forderung referenzierte Forderungssignatur wurde innerhalb der Gültigkeitsdauer der Signaturberechtigung erstellt.	C2PA-Anspruchsignaturfeld
`claimSignature.validated`	Die im Anspruch referenzierte Anspruchsignatur wurde validiert.	C2PA-Anspruchsignaturfeld
`Zutat.Anspruchsignatur.validiert`	Der Hash der C2PA-Anspruchsignaturbox des Inhaltsstoffs wurde erfolgreich validiert.	C2PA-Assertion

Wert	Bedeutung	`url` Verwendung
`Zutatenliste.manifest.validiert`	Der Hash des C2PA-Manifestfelds des Inhaltsstoffs wurde erfolgreich validiert.	C2PA-Assertion
`signingCredential.ocsp.notRev oked`	Die Signaturberechtigung war zum Zeitpunkt der Signatur nicht widerrufen.	C2PA-Anspruchsignaturfeld
`signingCredential.trusted`	Die Signaturberechtigung ist vertrauenswürdig.	C2PA-Anspruchsignaturfeld
`timeStamp.trusted`	Die Zeitstempel-Berechtigungsnachweis ist in der Liste der Vertrauensanker für Zeitstempelbehörden des Validators aufgeführt.	C2PA-Anspruchsignaturfeld
`timeStamp.validated`	Der Zeitstempel ist korrekt formatiert, verfügt über einen Nachrichtenabdruck, der mit der Anspruchsignatur übereinstimmt, und wurde innerhalb der Gültigkeitsdauer der Zeitstempel-Anmeldeinformationen erstellt.	C2PA-Anspruchsignaturfeld

15.2.2.2. Informationscodes

Tabelle 3. Validierungsinformationscodes

Wert	Bedeutung	`url` Verwendung
`algorithm.deprecated`	Der Algorithmus ist veraltet.	C2PA-Anspruchsfeld oder C2PA-Behauptung
`Zutat.unbekannteHerkunft`	Der Inhaltsstoff verfügt über kein C2PA-Manifest.	C2PA-Assertion
`signingCredential.ocsp.inacce ssible`	Der Validator hat versucht, eine Online-OCSP-Prüfung durchzuführen, aber keine Antwort erhalten.	C2PA-Anspruchsignaturfeld
`signingCredential.ocsp.skippe d`	Der Validator hat sich entschieden, keine Online-OCSP-Prüfung durchzuführen.	C2PA-Anspruchsignaturfeld
`timeOfSigning.insideValidity`	Der angegebene Zeitpunkt der Unterzeichnung (im IAT-Header der Signatur) liegt innerhalb der Gültigkeitsdauer der Zertifikatskette des Unterzeichners und vor dem Zeitpunkt eines entsprechenden vertrauenswürdigen Zeitstempels.	C2PA-Signaturfeld
`timeOfSigning.outsideValidity`	Der angegebene Zeitpunkt der Signatur (im iat-Header der Signatur) liegt außerhalb des Gültigkeitszeitraums der Zertifikatskette des Unterzeichners oder nach dem Zeitpunkt eines entsprechenden vertrauenswürdigen Zeitstempels.	C2PA-Anspruchsignaturfeld
`timeStamp.malformed`	Die im Claim-Signatur-Header enthaltene Zeitstempelantwort ist gemäß RFC 3161 nicht korrekt formatiert.	C2PA-Anspruchsignaturfeld

Wert	Bedeutung	`url` Verwendung
`timeStamp.mismatch`	Der Zeitstempel stimmt nicht mit dem Inhalt des Anspruchs überein.	C2PA-Anspruchsignaturfeld
`timeStamp.outsideValidity`	Das signierte Zeitstempelattribut in der Signatur wurde außerhalb der Gültigkeitsdauer des TSA-Zertifikats erstellt.	C2PA-Anspruchsignaturfeld
`timeStamp.untrusted`	Die Zeitstempel-Berechtigungsnachweis ist nicht in den TSA-Vertrauenslisten des Validators aufgeführt.	C2PA-Anspruchsignaturfeld

15.2.2.3. Fehlercodes

Tabelle 4. Validierungsfehlercodes

Wert	Bedeutung	`url` Verwendung
`algorithm.unsupported`	Der Algorithmus ist nicht spezifiziert oder wird nicht unterstützt.	C2PA-Anspruchsfeld oder C2PA-Behauptung
`assertion.action.ingredientMi smatch`	Eine Aktion, die eine zugehörige Zutat erfordert, hat entweder keine oder die angegebene kann nicht gefunden werden.	C2PA-Assertion
`assertion.action.malformed`	Eine Aktionsaussage ist fehlerhaft.	C2PA-Assertion
`assertion.action.redacted`	Eine Aktionsaussage wurde bei der Erstellung des Anspruchs redigiert.	C2PA-Assertion
`assertion.action.redactionMis match`	Eine Aktion, die eine zugehörige Redaktion erfordert, hat entweder keine oder die angegebene kann nicht gefunden werden.	C2PA-Assertion
`assertion.bmffHash.malformed`	Eine BMFF-Hash-Assertion ist fehlerhaft.	C2PA-Assertion
`assertion.bmffHash.mismatch`	Der Hash eines boxbasierten Assets stimmt nicht mit dem in einer BMFF-Hash-Assertion angegebenen Hash überein.	C2PA-Assertion
`assertion.boxesHash.malformed`	Die allgemeine Box-Hash-Assertion ist fehlerhaft.	C2PA-Assertion
`assertion.boxesHash.mismatch`	Der Hash eines allgemeinen boxähnlichen Asset-Formats stimmt nicht mit dem in einer allgemeinen Box-Hash-Assertion deklarierten Hash überein.	C2PA-Assertion
`assertion.boxesHash.unknownBo x`	Es wurde eine andere Box als erwartet gefunden.	C2PA-Assertion
`assertion.cloud-data.actions`	Ein Aktualisierungsmanifest enthält eine Cloud-Daten-Assertion, die auf eine Aktions-Assertion verweist.	C2PA-Assertion
`assertion.cloud-data.hardBinding`	Eine Hard-Binding-Assertion befindet sich in einer Cloud-Daten-Assertion.	C2PA-Assertion

Wert	Bedeutung	Verwendung der URL
Behauptung.Cloud-Daten.Fehlerhaft	Die Cloud-Daten-Assertion war unvollständig	C2PA-Assertion
assertion.cbor.invalid	Das Cbor einer Assertion ist ungültig	C2PA-Assertion
assertion.collectionHash.incorrectFileCount	Ein Asset, das in der Assertion zum Daten-Hash der Sammlung aufgeführt war, fehlt in der Sammlung.	C2PA-Assertion
assertion.collectionHash.invalidURI	Die URI eines Assets in der Hash-Assertion der Sammlungsdaten enthält den Dateiteil „..“ oder „.“.	C2PA-Assertion
assertion.collectionHash.malformed	Die Sammlungs-Hash-Assertion war unvollständig.	C2PA-Assertion
assertion.collectionHash.mismatch	Der Hash eines Assets in der Sammlung stimmt nicht mit dem in der Hash-Assertion der Sammlungsdaten angegebenen Hash überein.	C2PA-Assertion
assertion.dataHash.malformed	Eine Daten-Hash-Assertion ist fehlerhaft.	C2PA-Assertion
assertion.dataHash.mismatch	Der Hash eines Bytebereichs des Assets stimmt nicht mit dem in der Daten-Hash-Assertion deklarierten Hash überein.	C2PA-Assertion
assertion.dataHash.redacted	Eine Hard-Binding-Assertion wurde bei der Erstellung des Anspruchs redigiert.	C2PA-Assertion
assertion.hashedURI.mismatch	Der Hash der referenzierten Assertion im Manifest stimmt nicht mit dem entsprechenden Hash in der gehashten URI der Assertion im Anspruch überein.	C2PA-Assertion
assertion.inaccessible	Eine nicht eingebettete (remote) Assertion war zum Zeitpunkt der Validierung nicht zugänglich.	C2PA-Assertion
assertion.ingredient.malformed	Die Assertion zum Inhaltsstoff war unvollständig.	C2PA-Assertion
assertion.json.invalid	Das JSON(-LD) einer Assertion ist ungültig	C2PA-Aussage
assertion.missing	Eine im Manifest aufgeführte Assertion fehlt im Manifest des Assets.	C2PA-Anspruchsfeld
assertion.multiAssetHash.malformed	Eine Multi-Asset-Hash-Assertion ist fehlerhaft.	C2PA-Assertion
assertion.multiAssetHash.missingPart	Ein erforderlicher Teil des mehrteiligen Assets kann nicht gefunden werden.	C2PA-Assertion
assertion.multiAssetHash.mismatch	Der Hash eines Teils eines mehrteiligen Assets stimmt nicht mit dem in der Multi-Asset-Hash-Assertion deklarierten Hash überein.	C2PA-Assertion
assertion.multipleHardBindings	Das Manifest enthält mehr als eine Hard-Binding-Assertion.	C2PA-Assertion Store Box

Wert	Bedeutung	`url` Verwendung
`assertion.notRedacted`	Eine Assertion wurde in der Forderung als redigiert deklariert, ist aber weiterhin im Manifest vorhanden.	C2PA-Assertion
`assertion.outsideManifest`	Eine in der Forderung aufgeführte Behauptung befindet sich nicht im selben C2PA-Manifest wie die Forderung.	C2PA-Anspruchsfeld
`assertion.selfRedacted`	Eine Assertion wurde durch ihren eigenen Anspruch als redigiert deklariert.	C2PA-Anspruchsfeld
`assertion.timestamp.malformed`	Die Zeitstempel-Assertion ist fehlerhaft.	C2PA-Assertion
`assertion.undeclared`	Im Manifest wurde eine Behauptung gefunden, die in der Forderung nicht ausdrücklich angegeben war.	C2PA-Behauptung
`claim.cbor.invalid`	Das cbor der Forderung ist ungültig.	C2PA-Anspruchsbox
`claim.hardBindings.missing`	Es sind keine Hard Bindings vorhanden.	C2PA-Anspruchsbox
`claim.malformed`	Die Daten/Felder des referenzierten Anspruchs im Manifest sind nicht korrekt.	C2PA-Anspruchsbox
`claim.missing`	Der im Manifest referenzierte Anspruch kann nicht gefunden werden.	C2PA-Anspruchsfeld
`claim.multiple`	Das Manifest enthält mehr als eine Claim Box.	C2PA-Anspruchsfeld
`claimSignature.missing`	Die im Anspruch referenzierte Anspruchsignatur kann im Manifest nicht gefunden werden.	C2PA-Anspruchsignaturfeld
`claimSignature.mismatch`	Die im Anspruch angegebene Anspruchsignatur konnte nicht validiert werden.	C2PA-Anspruchsignaturfeld
`claimSignature.outsideValidit y`	Die im Anspruch angegebene Anspruchsignatur wurde außerhalb des Gültigkeitszeitraums der Signaturberechtigung erstellt.	C2PA-Anspruchsignaturfeld
`general.error`	Ein Wert, der verwendet wird, wenn ein Fehler aufgetreten ist, der hier nicht ausdrücklich aufgeführt ist.	C2PA-Anspruchsfeld oder C2PA-Behauptung
`hashedURI.missing`	Die Daten, auf die ein `hashed_uri` verweist, können nicht gefunden werden.	C2PA-Assertion
`hashedURI.mismatch`	Der Hash einer bestimmten `hashed_uri` stimmt nicht mit dem entsprechenden Hash der Daten der Ziel-URI überein	C2PA-Assertion
`ingredient.claimSignature.mis sing`	Die referenzierte C2PA-Signatur für die Zutat wurde nicht gefunden.	C2PA-Assertion

Wert	Bedeutung	`url` Verwendung
`Zutat.claimSignature.mis Übereinstimmung`	Der Hashwert der C2PA-Anspruchsignatur eines eingebetteten C2PA-Manifests stimmt nicht mit dem Hashwert überein, der im Wert `„hashed_uri"` des Feldes `„claimSignature"` in der Zutatenangabe angegeben ist.	C2PA-Assertion
`Zutat.Manifest fehlt`	Das referenzierte C2PA-Manifest der Zutat wurde nicht gefunden.	C2PA-Assertion
`Zutat.Manifest.Nichtübereinstimmung`	Der Hash eines eingebetteten C2PA-Manifests stimmt nicht mit dem Hash überein, der im Wert `„hashed_uri"` des Felds `„activeManifest"` in der Zutaten-Assertion angegeben ist.	C2PA-Assertion
`manifest.compressed.invalid`	Das komprimierte Manifest war ungültig.	C2PA-Anspruchsfeld
`manifest.inaccessible`	Ein nicht eingebettetes (remotes) Manifest war zum Zeitpunkt der Validierung nicht zugänglich.	C2PA-Anspruchsbox
`manifest.multipleParents`	Das Manifest enthält mehr als eine Zutat, deren `Beziehung „parentOf"` lautet.	C2PA-Anspruchsfeld
`manifest.timestamp.invalid`	Das Manifest ist ein Zeitstempel-Manifest, enthält jedoch eine unzulässige (nicht zum Bestandteil gehörende) Assertion.	C2PA-Anspruchsfeld
`manifest.timestamp.wrongParen ts`	Das Manifest ist ein Zeitstempel-Manifest, enthält jedoch entweder null oder mehrere `parentOf`-Inhaltsstoffe.	C2PA-Anspruchsfeld
`manifest.update.invalid`	Das Manifest ist ein Aktualisierungsmanifest, enthält jedoch eine unzulässige Assertion, z. B. eine Hard-Binding- oder Aktions-Assertion.	C2PA-Anspruchsfeld
`manifest.update.wrongParents`	Das Manifest ist ein Aktualisierungsmanifest, enthält jedoch entweder null oder mehrere `parentOf`-Bestandteile.	C2PA-Anspruchsfeld
`signingCredential.invalid`	Die Signaturberechtigung ist für die Signatur ungültig.	C2PA-Anspruchsignaturfeld
`signingCredential.ocsp.revoke d`	Die OCSP-Antwort zeigt an, dass die Signaturberechtigung vom Aussteller widerrufen wurde.	C2PA-Anspruchsignaturfeld
`signingCredential.ocsp.unknow n`	Die OCSP-Antwort enthält einen `unbekannten` Status für die Signaturberechtigung.	C2PA-Anspruchsignaturfeld
`signingCredential.untrusted`	Die Signaturberechtigung ist in keiner der geltenden Vertrauenslisten des Validators aufgeführt.	C2PA-Anspruchsignaturfeld

15.3. Anzeige von Manifestinformationen

Manifest-Konsumenten sollten keine Daten aus Manifesten anzeigen, die nicht gültig sind, oder aus Assets, die nicht gültig sind. Wenn der Manifest-Konsument sich dafür entscheidet, solche Daten anzuzeigen, muss er als Teil der Anzeige Folgendes einfügen:

- eine Warnung über die mangelnde Gültigkeit,
- eine Warnung, dass die Daten nicht dem Unterzeichner des Manifests zugeordnet werden dürfen, und im Falle eines Inhaltsstoffmanifests zusätzlich nicht dem Unterzeichner des Manifests des Vermögenswerts.

HINWEIS Beim Verfassen von Szenarien ist es wünschenswert, Warnungen deutlicher hervorzuheben, damit der Ersteller eine fundierte Entscheidung darüber treffen kann, wie mit einem Asset verfahren werden soll, das nicht gültig ist oder eine fehlerhafte Herkunftshistorie aufweist .

15.4. Festlegung des Hash-Algorithmus

15.4.1. Für gehasht

Verschiedene Teile des C2PA-Manifests verwenden eine hashed_uri-Struktur, um eine URI, ihren Hash und (optional) den zur Berechnung des Hash verwendeten Algorithmus zu kapseln. Wenn die hashed_uri-Struktur ein alg-Feld enthält, wird dieses als Hash-Algorithmus verwendet. Ist das alg-Feld in der hashed_uri-Struktur nicht vorhanden, wird der Hash-Algorithmus durch Auswertung der nächstgelegenen umschließenden Struktur bestimmt, die ein alg-Feld enthält. Wenn an keiner dieser Stellen ein `alg`-Feld gefunden wird, muss der Wert des alg-Feldes im Anspruch als Hash-Algorithmus verwendet werden. Wenn an keiner dieser Stellen ein alg-Feld vorhanden ist, muss der Anspruch mit dem Fehlercode `algorithm.unsupported` abgelehnt werden.

15.4.2. Für gehasht Ext-URIs

Einige Teile eines C2PA-Manifests verwenden eine hashed_ext_uri-Struktur, um eine externe URI, ihren Hash und den zur Berechnung des Hash verwendeten Algorithmus zu kapseln. Wenn in der hashed_ext_uri-Struktur ein alg-Feld vorhanden ist, wird dieses als Hash-Algorithmus verwendet. Ist das alg-Feld in der hashed_ext_uri-Struktur nicht vorhanden, wird der Fehlercode `algorithm.unsupported` verwendet.

HINWEIS Das Feld „`alg`“ ist in „`hashed_ext_uri`“ obligatorisch, sodass kein rekursives Verfahren zur Bestimmung des Hash-Algorithmus erforderlich ist.

15.4.3. Algorithmusvalidierung

Sobald der Hash-Algorithmus festgelegt ist, muss er mit den Werten in der Liste der zulässigen Algorithmen oder der Liste der veralteten Algorithmen in Abschnitt 13.1, „Hashing“, verglichen werden. Ist er in keiner der beiden Listen enthalten, muss der Anspruch mit dem Fehlercode „`algorithm.unsupported`“ abgelehnt werden. Ist der Algorithmus in der Liste der veralteten Algorithmen enthalten, muss der Anspruch mit dem Informationscode „`algorithm.deprecated`“ versehen werden.

15.5. Suchen des aktiven Manifests

15.5.1. Allgemeines

Die letzte C2PA-Manifest-Superbox in der C2PA-Manifest-Store-Superbox gilt als aktives Manifest, aber das Auffinden des C2PA-Manifest-Stores kann die Suche an mehreren möglichen Orten erfordern.

15.5.2. Eingebettet

15.5.2.1. Allgemeines

Der C2PA-Manifest-Speicher muss vom Validator, der in das Asset eingebettet ist, an den Standardorten für die Einbettung von Manifesten gefunden werden. Wenn ein Asset jedoch über eine HTTP-Verbindung abgerufen wurde, kann ein Validator nach einem Link-Header suchen, wie in der folgenden Klausel „Link-Header" beschrieben, um festzustellen, ob ein C2PA-Manifest-Speicher vorhanden ist.

NOTE Durch Überprüfen des `Link`-Headers, sofern vorhanden, kann ein Validator feststellen, ob ein C2PA Manifest Store vorhanden ist, ohne das gesamte Asset herunterladen zu müssen. Dies ist nützlich für große oder gestreamte Assets.

Wenn mehrere C2PA-Manifest-Speicher in einem Asset vorhanden sind, werden sie alle als ungültig betrachtet, und die Validierung sollte so vorgehen, als ob keine Manifeste gefunden worden wären. Wenn dieses Asset als Bestandteil hinzugefügt wird, darf keines dieser eingebetteten C2PA-Manifeste in die Bestandteilangabe aufgenommen werden.

15.5.2.2. Besondere Überlegungen zu PDF

PDF-Dateien unterstützen eine Technologie namens „inkrementelle Aktualisierung", bei der Informationen an das Ende des Dokuments angehängt werden, anstatt das Original zu ändern. Dies erfordert, dass PDF-Dateien mehrere C2PA-Manifest-Speicher unterstützen – allerdings darf es nur einen pro Aktualisierungsabschnitt geben.

Wenn in einem einzelnen Aktualisierungsabschnitt mehrere C2PA-Manifest-Speicher vorhanden sind, werden sie alle als ungültig betrachtet, und die Validierung sollte so erfolgen, als ob keine Manifeste gefunden worden wären. Alle C2PA-Manifest-Speicher, die in früheren Aktualisierungen der PDF-Datei oder der Original-PDF-Datei vorhanden sind, gelten jedoch weiterhin als gültig und werden entsprechend verarbeitet.

15.5.3. Durch Verweis oder URI

15.5.3.1. Durch Referenz

Wenn kein C2PA-Manifest-Speicher eingebettet ist, sollten die folgenden Versuche unternommen werden, um einen an einem entfernten Speicherort zu finden.

- Wenn das Asset über eine HTTP-Verbindung abgerufen wurde, beschreibt die folgende Link-Header-Klausel, wie ein Manifest über den `Link`-Header gefunden werden kann.
- Wenn das Asset über XMP an den Standard-Asset-Speicherorten verfügt (d. h. außerhalb des C2PA-Manifests) und dieses XMP einen `dcterms:provenance`-Schlüssel enthält, sollte die angegebene URI verwendet werden, um das aktive Manifest zu finden.

- Wenn es sich bei dem Asset um eine Schriftart mit einer C2PA-Tabelle handelt und dessen `activeManifestUriLength` ungleich Null ist, sollte die angegebene URI verwendet werden, um das aktive Manifest zu finden.
- Wenn kein C2PA-Manifest-Speicher gefunden wurde, sollte der Validator nach Dateien mit demselben Pfad oder derselben URI suchen, jedoch mit der Dateinamenerweiterung `.c2pa`. Wenn der C2PA-Manifest-Speicher nicht gefunden wird, kann ein Validator an allen weiteren Orten suchen, die er für am besten geeignet hält, um ihn zu finden. Zum Beispiel in einem Unterordner eines Dateisystems.

HINWEIS	Ein Validator ist nicht auf die oben genannten Speicherorte beschränkt, sondern kann auch an weiteren Speicherorten suchen.

Wenn ein Manifest an einem entfernten Speicherort dokumentiert wurde, dort jedoch nicht vorhanden ist oder der Speicherort derzeit nicht verfügbar ist (z. B. in einem Offline-Szenario), muss der Fehlercode „`manifest.inaccessible`“ verwendet werden, um die Situation zu melden.

Informationen zum IANA-Medientyp für einen C2PA-Manifest-Speicher finden Sie im Abschnitt „Externe Manifeste“.

15.5.3.2. Über `den Link`-Header

Wenn das Asset über eine HTTP-Verbindung abgerufen wurde, sollte der Validator im Header der HTTP-Antwort nach einem Link-Header suchen, wie in RFC 8288 definiert, der einen Parameter `rel=c2pa-manifest` enthält. Ist dieser vorhanden, kann ein C2PA-Manifest-Speicher aus dieser URI-Referenz abgerufen werden. Die URI ist eine `Standard-HTTP-` oder HTTPS-URI, z. B. https://c2pa.org/image.c2pa.

Es ist auch möglich, die Link-Relation zu verwenden, um auf den C2PA-Manifest-Speicher zu verweisen, der durch die Verwendung eines JUMBF-URI-Fragments in ein Asset eingebettet ist. Die URI würde ein JUMBF-URI-Fragment zum C2PA-Manifest-Speicher-Superbox https://c2pa.org/image.jpg#jumbf=c2pa enthalten. Verweise auf bestimmte C2PA-Manifeste innerhalb des C2PA-Manifest-Speichers sind nicht zulässig, und der Validator muss alle `Childlabel`-Teile des JUMBF-URI-Fragments ignorieren.

HINWEIS	HTTP bezieht sich auf das in RFC 7230 definierte *Hypertext Transfer Protocol*, nicht auf das spezifische URL-Schema `http://`

15.5.4. Dekomprimierung

Wie zuvor beschrieben, können sowohl Standard- als auch Aktualisierungsmanifeste komprimiert werden. Wenn ein komprimiertes Manifest gefunden wird, muss ein Validator es dekomprimieren, bevor er mit dem Standardvalidierungsprozess fortfährt. Wenn die in der Brob-Box eines komprimierten Manifests enthaltenen Daten weder ein Standard- noch ein Aktualisierungsmanifest sind oder wenn die Dekomprimierung fehlschlägt, muss der Validator das Manifest mit dem Fehlercode `manifest.compressed.invalid` ablehnen.

15.5.5. Validierung einer Übereinstimmung

Ein Validierer möchte möglicherweise überprüfen, ob der gefundene C2PA-Manifest-Speicher tatsächlich derjenige ist, der mit dem Asset verknüpft ist.

Wenn der C2PA-Manifest-Speicher gefunden wurde, muss anhand der in seinem aktiven Manifest enthaltenen Hard-Binding-Assertion überprüft werden, ob es sich um das passende Manifest handelt und ob das Asset ohne Manifest-Aktualisierungen geändert wurde. Wenn das Hard-Binding nicht übereinstimmt, ist unklar, ob dies auf (a) eine Änderung des Assets oder (b) einen falschen C2PA

Manifest Store gefunden wurde. Dementsprechend muss der Validierer dies als nicht übereinstimmende harte Bindung behandeln und das Manifest mit einem Fehlercode von `assertion.dataHash.mismatch` ablehnen, wenn eine Daten-Hash-Assertion verwendet wird, von `assertion.boxesHash.mismatch`, wenn eine allgemeine Box-Hash-Assertion verwendet wird, von `assertion.collectionHash.mismatch`, wenn eine Sammlungsdaten-Hash-Assertion verwendet wird, oder von `assertion.bmffHash.mismatch`, wenn eine BMFF-Hash-Assertion verwendet wird.

15.6. Lokalisieren und Validieren des Anspruchs

15.6.1. Lokalisieren

Sobald das zu validierende Manifest gefunden wurde (im Folgenden als „aktuelles Manifest" bezeichnet), wird der Anspruch gefunden, indem innerhalb des aktuellen Manifests die JUMBF-Superbox mit der Bezeichnung `c2pa.claim.v2` (oder `c2pa.claim` für Dateien mit älteren Anspruchsstrukturen) und einer JUMBF-Typ-UUID von `6332636C-0011-0010-8000-00AA00389B71` (`c2cl`) gefunden. Beachten Sie, dass die UUID vom Typ JUMBF sowohl für das neue (mit der Bezeichnung `c2pa.claim.v2`) als auch für das alte (mit der Bezeichnung `c2pa.claim`) Anspruchsformat identisch ist. Es darf nur ein solches Feld im aktuellen Manifest vorhanden sein. Wenn mehr als eines gefunden wird, wird das C2PA-Manifest mit dem Fehlercode `claim.multiple` abgelehnt.

15.6.2. Validierung

Wenn der Inhalt der Forderung kein wohlgeformtes CBOR ist, wird die Forderung mit dem Fehlercode
c`laim.cbor.invalid`

HINWEIS Ein korrekt formatiertes CBOR ist in RFC 8949, Anhang C definiert.

Für einen „c2pa.claim.v2" müssen die folgenden Felder im CBOR-Objekt vorhanden sein. Fehlt eines davon, wird der Anspruch mit dem Fehlercode `claim.malformed` abgelehnt.

- `instanceID`
- `signature`
- `created_assertions`
- `claim_generator_info`

Wenn das Feld `claim_generator_info` kein Feld `name` enthält, wird der Anspruch mit dem Fehlercode `claim.malformed` abgelehnt.

Wenn es in der `generator-info-map` ein Icon-Feld gibt, auf das das Feld `claim_generator_info` der `claim-map` oder `claim-map-v2` verweist, muss dessen Wert gemäß Abschnitt 15.10.3.3, „Validierung von Verweisen", validiert werden.

15.7. Signatur validieren

Rufen Sie die URI-Referenz für die Signatur aus dem Wert des Signaturfelds des Anspruchs ab und lösen Sie die URI auf.

Verweis zum Abrufen der COSE-Signatur. Die Signatur muss in dasselbe Manifest eingebettet sein, wie in Abschnitt 11.1.4, „C2PA-Box-Details“, beschrieben. Wenn die Signatur-URI nicht auf einen Ort innerhalb derselben C2PA-Manifest-Box (ein self#jumbf-Ort) verweist, wird der Anspruch abgelehnt. Wenn kein solches Feld vorhanden ist oder die URI nicht aufgelöst werden kann, wird der Anspruch mit dem Fehlercode `claimSignature.missing` abgelehnt.

Wenn die Signatur und der Anspruch nicht im selben C2PA-Manifest enthalten sind, gilt dieses C2PA-Manifest als ungültig.

Für alle Arten von C2PA-Manifesten muss die Validierung der in der Signatur verwendeten Berechtigungsnachweise gemäß Kapitel 14, *Vertrauensmodell*, durchgeführt werden.

Wenn die Berechtigungsnachweis gemäß den Anforderungen des Berechtigungsnachweis-Typs nicht akzeptabel ist, muss der Anspruch mit dem Fehlercode `„signingCredential.invalid“` abgelehnt werden. Wenn der Signaturalgorithmus nicht in der Liste der zulässigen oder veralteten Algorithmen in Abschnitt 13.2, „Digitale Signaturen“, aufgeführt ist, muss der Anspruch mit dem Fehlercode `„algorithm.unsupported“` abgelehnt werden.

Anschließend muss eine Vertrauenskette von der Berechtigungsnachweis zu einem Eintrag in einer der geltenden Vertrauensankerlisten überprüft werden. Wenn diese Vertrauenskette nicht überprüft werden kann, wird der Anspruch mit dem Fehlercode `„signingCredential.untrusted“` abgelehnt; andernfalls wird der Anspruchsignatur der Erfolgscode `„signingCredential.trusted“` zugewiesen.

Wenn der Anspruch noch nicht abgelehnt wurde, wird die Validierung gemäß dem in Abschnitt 13.2, „Digitale Signaturen“, angegebenen Verfahren fortgesetzt. Wenn die Validierung der Signatur fehlschlägt, wird der Anspruch mit dem Fehlercode `claimSignature.mismatch` abgelehnt. Andernfalls wird der Anspruchsignatur der Erfolgscode `claimSignature.validated` zugewiesen.

Im weiteren Verlauf dieses Kapitels beziehen sich Header auf die Vereinigung der geschützten und ungeschützten Header-Parameter in der COSE-Signatur. Sofern in Abschnitt 13.2, „Digitale Signaturen“, oder Abschnitt 14.5, „X.509-Zertifikate“, nicht anders angegeben, kann ein Header in beiden Gruppen vorkommen. COSE-Header werden in RFC 8152, Abschnitt 3, beschrieben.

15.8. Zeitstempel validieren

15.8.1. Abrufen des Zeitstempel-Tokens

15.8.1.1. Eingebettet in die Anspruchsignatur

Wenn entweder der `sigTst-` oder der `sigTst2`-Header vorhanden ist, wird erwartet, dass das tstTokens-Array ein einzelnes `tstToken` enthält. Wenn der Header mehr als ein `tstToken` enthält, gibt der Validator einen Informationscode `„timestamp.malformed“` aus und ignoriert die Zeitstempel.

Ein Validator, der `sigTst` unterstützt, muss die folgenden Verfahren durchführen, um die Zeitstempelantwort zu validieren:

- Rufen Sie die Eigenschaft `„val”` aus dem `tstToken` ab, bei dem es sich um eine RFC3161-konforme `TimeStampResp` (Zeitstempelantwort) handeln muss.

- Überprüfen Sie den Wert des Statusfelds `PKIStatusInfo`, bei dem es sich um den Wert des Statusfelds von `TimeStampResp`.
    - Wenn es einen anderen Wert als `0` (`gewährt`) oder `1` (`gewährtMitModifikationen`) enthält, muss der Validator eine `timeStamp.malformed` Informationscode und ignorieren Sie diesen Zeitstempel.
    - Wenn er entweder `0` (`gewährt`) oder `1` (`gewährtMitÄnderungen`) ist, fahren Sie mit dem Rest des Zeitstempel-Validierungsprozesses fort, wie unten beschrieben.
- Rufen Sie den Wert des Feldes „`timeStampToken`" der „`TimeStampResp`" ab, um ihn im weiteren Verlauf des Validierungsprozesses zu verwenden.

Ein Validator für `sigTst2` muss die Eigenschaft „`val`" aus dem `tstToken` abrufen, bei dem es sich um einen RFC3161-konformen `timeStampToken` (TimeStampToken, TST) sein.

15.8.1.2. Referenziert durch eine Zeitstempel-Assertion

Wenn ein Validierer bereits ein TimeStampToken in einem `sigTst`- oder sigTst2-Header gefunden hat, das die Validierung besteht (gemäß Abschnitt 15.8.2, „Validieren des TimeStampTokens"), muss er diesen Schritt überspringen. Wenn kein solcher Header vorhanden ist oder das dort gefundene TimeStampToken die Validierung nicht bestanden hat, muss dieser Schritt durchgeführt werden.

Wenn ein Validierer zuvor Zeitstempel-Assertions gefunden hat, die dann in einer Zuordnung von C2PA-Manifest-Identifikatoren zu TimeStampTokens gespeichert wurden, muss der Validierer überprüfen, ob der Identifikator des aktuellen C2PA-Manifests in der Zuordnung vorhanden ist. Ist dies der Fall, verwendet der Validator das mit der Kennung in der Zuordnung verknüpfte TimeStampToken für das TimeStampToken im Validierungsprozess, der in Abschnitt 15.8.2, „Validieren des TimeStampTokens", beschrieben ist. Wenn in der Zuordnung mehr als ein TimeStampToken für diese Kennung gefunden wird, muss der Validierer jedes einzelne ausprobieren, bis eines die Validierung erfolgreich durchläuft (und dann die anderen ignorieren). Wenn die Kennung nicht in der Zuordnung erscheint, wird kein Fehler ausgelöst, da dies lediglich bedeutet, dass es im aktuellen Kontext kein mit diesem C2PA-Manifest verknüpftes TimeStampToken gibt.

15.8.2. Validierung des TimeStampToken

Alle Validatoren müssen den Prozess wie folgt fortsetzen:

- Wenn der Signaturalgorithmus im `timeStampToken` nicht in der Liste der zulässigen oder veralteten Algorithmen in Abschnitt 13.2, „Digitale Signaturen", aufgeführt ist, muss der Validierer einen Informationscode „`timestamp.untrusted`" ausgeben und den Zeitstempel ignorieren.
- Validieren Sie die Signatur im `timeStampToken` gemäß der Beschreibung in RFC 2630, Abschnitt 5.6. Wenn die Signatur nicht gültig ist, muss der Validator einen Informationscode „`timestamp.mismatch`" ausgeben und den Zeitstempel ignorieren.
- Wenn das `timeStampToken` nicht enthält ein messageImprint-Feld, muss der Validator einen `timestamp.malformed` Informationscode ausgeben und den Zeitstempel ignorieren.
- Wenn der Hash-Algorithmus für den Nachrichten-Imprint nicht in der Liste der zulässigen oder veralteten Algorithmen in Abschnitt 13.1, „Hashing", aufgeführt ist, muss der Validator einen Informationscode „`timestamp.untrusted`" ausgeben und den Zeitstempel ignorieren.
- Überprüfen Sie, ob der Wert des Feldes `messageImprint` (im `timeStampToken`) entweder mit dem Anspruch (v1,

sigTst) oder das Signaturfeld der COSE_Sign1_Tagged-Struktur (v2, sigTst2) des zu validierenden C2PA-Manifests, wie in Abschnitt 10.3.2.5.2, „Auswahl der Nutzlast“, beschrieben. Stimmen die Werte nicht überein, gibt der Validator einen Informationscode „timestamp.mismatch“ aus und ignoriert den Zeitstempel.

- Es ist zu überprüfen, ob das Zertifikatsfeld des timeStampToken vorhanden ist, ob das TSA-Zertifikat in dem in diesem Feld bereitgestellten Zertifikatssatz zu finden ist und ob es möglich ist, eine Vertrauenskette vom TSA-Zertifikat zu einem Eintrag in der C2PA TSA Trust List (oder einer anderen Liste von Vertrauensankern, die zu diesem Zweck im Validator vorhanden ist) aufzubauen. Wenn das Zertifikat nicht gefunden werden kann oder keine Vertrauenskette aufgebaut werden kann, gibt der Validator einen Informationscode „timestamp.untrusted“ aus und ignoriert den Zeitstempel.
- Überprüfen Sie, ob die im Feld „genTime“ (im „timeStampToken“) angegebene beglaubigte Zeit innerhalb der Gültigkeitsdauer des Signaturzertifikats der TSA und aller CA-Zertifikate bis zum Vertrauensanker liegt. Ist dies nicht der Fall, muss der Validierer einen Informationscode „timestamp.outsideValidity“ ausgeben und den Zeitstempel ignorieren.
- Wenn die Zeitstempelvalidierung aufgrund einer der oben genannten Bedingungen nicht unterbrochen wird oder fehlschlägt, gibt der Validator die Erfolgscodes „timeStamp.trusted“ und „timeStamp.validated“ aus.
- Wenn der Validierer sowohl den Erfolgscode „timeStamp.trusted“ als auch den Erfolgscode „timeStamp.validated“ ausgegeben hat, muss der Validierer überprüfen, ob die von der Zeitstempelbehörde (TSA) beglaubigte Zeit, die im Feld „genTime“ (im „timeStampToken“) zu finden ist, innerhalb der Gültigkeitsdauer des Zertifikats zur Signatur des Anspruchs und aller CA-Zertifikate bis zum Vertrauensanker liegt. Ist dies nicht der Fall, muss der Validierer den Anspruch mit dem Fehlercode claimSignature.outsideValidity ablehnen.

HINWEIS Zeitstempel bleiben auch nach Ablauf der Signaturberechtigung der Zeitstempelstelle gültig, daher , solange der beglaubigte Zeitpunkt innerhalb der Gültigkeitsdauer des Zertifikats der Zeitstempelstelle liegt. Dies ist eine besondere Art des Vertrauens, die nur Zeitstempelstellen gewährt wird.

Zum Zeitpunkt der Validierung, wenn ein Zeitstempel vorhanden, vertrauenswürdig und validiert ist, müssen Validierer die beglaubigte Zeit und nicht die aktuelle Zeit verwenden, um die zeitliche Gültigkeit des Signaturzertifikats und des Zertifikats der Zeitstempelstelle zu bestimmen.

HINWEIS Dieses Dokument verlangt nicht, dass der Widerrufsstatus des Zertifikats einer Zeitstempelstelle zum Zeitpunkt der Signatur erfasst oder zum Zeitpunkt der Validierung validiert wird.

Wenn weder der sigTst- noch der sigTst2-Header vorhanden sind oder wenn mindestens einer von ihnen vorhanden ist, aber ihr Zeitstempel-Token die oben genannten Anforderungen nicht erfüllt, ist das C2PA-Manifest gültig, wenn die aktuelle Zeit bei der Validierung innerhalb der Gültigkeitsdauer des Zertifikats des Unterzeichners und aller CA-Zertifikate bis zum Vertrauensanker liegt. Ist dies der Fall, gibt der Validierer einen Erfolgscode von claimSignature.insideValidity zurück. Ist dies nicht der Fall, wird das C2PA-Manifest mit einem Fehlercode von claimSignature.outsideValidity abgelehnt.

15.8.3. Validierung der „angegebenen Zeit der Unterzeichnung”

Ein Validierer kann sich dafür entscheiden, den „angegebenen Zeitpunkt der Unterzeichnung“ zu validieren, wie er durch den Wert im geschützten iat-Header bestätigt wird. Wenn der iat-Header vorhanden ist, kann der Validierer überprüfen, ob der bestätigte Zeitpunkt innerhalb der Gültigkeitsdauer des Zertifikats des Unterzeichners und aller CA-Zertifikate bis zum Vertrauensanker liegt und nicht später als der durch einen zugehörigen vertrauenswürdigen Zeitstempel bestätigte Zeitpunkt ist. Wenn der Validierer die Validierung dieses Werts durchführt und dieser innerhalb des Gültigkeitszeitraums liegt,

gibt der Validierer den Informationscode „timeOfSigning.insideValidity“ zurück. Liegt der Wert jedoch außerhalb des Gültigkeitszeitraums, gibt der Validierer den Informationscode „timeOfSigning.outsideValidity“ zurück.

15.9. Validieren Sie die Informationen zur Sperrung von Berechtigungsnachweisen

Der Validator muss versuchen, den Widerrufsstatus des Zertifikats des Unterzeichners und aller CA-Zertifikate, die Teil der Vertrauenskette sind, zu ermitteln.

Bei CA-Zertifikaten sollte der Validierer den Widerrufsstatus gemäß der Erweiterung „Authority Information Access“ (AIA) bestimmen, wie in RFC 5280, Abschnitt 4.2.2.1 beschrieben. Der Validierer sollte die entsprechenden OCSP-Antworten aus dem C2PA-Manifest verwenden, wenn die AIA-Erweiterung angibt, dass OCSP verfügbar ist.

Wenn der Validator feststellt, dass ein CA-Zertifikat zu dem in einem vertrauenswürdigen Zeitstempel angegebenen Zeitpunkt oder, falls kein vertrauenswürdiger Zeitstempel vorhanden ist, zum aktuellen Zeitpunkt widerrufen wurde, wird die Signatur der Behauptung mit dem Fehlerstatus „signingCredential.untrusted“ abgelehnt.

Für das Zertifikat des Unterzeichners muss der Validator das folgende Verfahren anwenden.

- Wenn ein Zertifikat den Widerrufsstatus nicht unterstützt oder der Zertifikatsaussteller keine Methode zur Abfrage seines Widerrufsstatus bereitgestellt hat, behandelt der Validator die Berechtigung als nicht widerrufen.
- Wenn der Anspruchgenerator OCSP-Antworten im rVals-Header der COSE_Sign1-Struktur „angeheftet” hat, muss der Validator die angehefteten OCSP-Antworten wie in Abschnitt 15.9.1, „Feststellung der Sperrung durch OCSP-Antworten im C2PA-Manifest-Speicher”, beschrieben decodieren und validieren.
- Wenn nachfolgende Anspruchgeneratoren Zertifikatsstatusaussagen in anderen C2PA-Manifesten im C2PA-Manifest-Speicher hinzugefügt haben, muss der Validierer diese OCSP-Antwort(en) im Validierungsprozess verwenden, wie in Abschnitt 15.9.1, „Feststellung der Sperrung durch OCSP-Antworten im C2PA-Manifest-Speicher“, beschrieben. Wenn mehr als eine OCSP-Antwort für das Zertifikat gefunden wird, muss der Validierer jede einzelne ausprobieren, bis eine die Validierung erfolgreich durchläuft (und dann die anderen ignorieren).

Wenn im C2PA Manifest Store keine Sperrinformationen gefunden wurden, der Validierer online ist und der Validierer den Sperrstatus für das Zertifikat überprüfen möchte, muss der Validierer versuchen, den Sperrstatus des Zertifikats zu ermitteln, indem er den OCSP-Responder wie in Abschnitt 15.9.2, „Ermittlung der Sperrung anhand der Online-OCSP-Antwort“, beschrieben abfragt.

15.9.1. Feststellung der Sperrung durch OCSP-Antworten im C2PA-Manifest-Speicher

Ein Validierer muss OCSP-Antworten gemäß den Anforderungen von RFC 6960, insbesondere den Anforderungen 1 bis 4 in Abschnitt 3.2, dekodieren. Wenn eine OCSP-Antwort akzeptiert wird und alle folgenden Anforderungen erfüllt sind, wird damit festgestellt, dass das betreffende Zertifikat zum Zeitpunkt der Signatur nicht widerrufen war.

- Die Signatur der Behauptung verfügt über eine beglaubigte Zeitangabe, die durch einen gültigen signierten Zeitstempel bereitgestellt wird.
- Es gibt eine SingleResponse im Array „responses“ des Feldes „tbsResponseData“ der OCSP-Antwort, sodass alle folgenden Bedingungen erfüllt sind:

- Die aktuelle Zeit ist nicht früher als `thisUpdate`.
- Die beglaubigte Zeit aus dem Zeitstempel:
  - ist früher als `thisUpdate` oder
  - liegt innerhalb des Intervalls `(thisUpdate,nextUpdate)`, wenn `nextUpdate` vorhanden ist, oder
  - liegt innerhalb des Intervalls `(thisUpdate, producedAt + 24 Stunden)`, wobei `producedAt` das Feld in den enthaltenen `ResponseData` ist, wenn `nextUpdate` nicht vorhanden ist.
- Das Feld `certStatus` der `SingleResponse` ist `gültig` oder `widerrufen`, jedoch mit einem `revocationReason` von `removeFromCRL`.

HINWEIS

Der `Wert „removeFromCRL"` ist unter den Werten von `„revocationReason"` einzigartig, da er einer `positiven` Antwort entspricht. Obwohl es sich um eine Art `widerrufene` Antwort handelt, ist diese Antwort , dass das Zertifikat zuvor aufgrund von Bedenken hinsichtlich seiner Integrität vorübergehend „gesperrt" worden war (Grund `„certificateHold"`), dass diese Bedenken jedoch ausgeräumt wurden und der Aussteller erklärt, dass das Zertifikat weiterhin vertrauenswürdig ist (siehe RFC 5280).

- Der OCSP-Signierer der Antwort ist ein „autorisierter Responder" gemäß der Definition in RFC 6960, Abschnitt 4.2.2.2.

Validatoren müssen den `revocationReason` jeder `widerrufenen` Antwort überprüfen, um den Fall `„removedFromCRL"` von einer tatsächlichen Sperrung unterscheiden zu können.

Wenn die oben genannten Bedingungen für eine OCSP-Antwort im C2PA-Manifest-Speicher erfüllt sind, gilt das Zertifikat zum Zeitpunkt der Signierung als nicht widerrufen, und der Validierer gibt den Erfolgscode `signingCredential.ocsp.notRevoked` aus.

Andernfalls erfüllt eine OCSP-Antwort im C2PA-Manifest-Speicher alle oben genannten Bedingungen mit Ausnahme des Feldes `„certStatus"`, das `widerrufen` ist, das Zertifikat gilt zum Zeitpunkt der Signatur als widerrufen und der Anspruch wird mit einem Fehlercode `„signingCredential.ocsp.revoked"` abgelehnt.

15.9.2. Feststellung der Sperrung anhand der Online-OCSP-Antwort

Wenn für ein bestimmtes Zertifikat keine OCSP-Antwort im C2PA Manifest Store die Bedingungen in Abschnitt 15.9.1 erfüllt, „Feststellung der Sperrung anhand von OCSP-Antworten im C2PA-Manifest-Speicher" erfüllt, oder wenn die Signatur der Behauptung keinen Zeitstempel aufweist, kann der Validierer gemäß RFC 6960 eine Abfrage an den OCSP-Responder senden, wobei die `accessLocation` des Responders gemäß RFC 6960, Abschnitt 3.1, ermittelt wird.

HINWEIS

Die Abfrage der Methode zum Status der Berechtigungsnachweise kann einem Beobachter die Identität des zu validierenden Assets offenbaren, daher ist diese Abfrage optional.

Wennder Validator entscheidet nicht , durchzuführen eine Online OCSP durchzuführen, wird soll ausgestellt einen `signingCredential.ocsp.skipped` Informationscode aus.

Wenn der Validator versucht, den OCSP-Responder abzufragen, aber keine Antwort erhält, muss der Validator einen `signingCredential.ocsp.inaccessible` Informationscode aus.

Wenn eine Antwort gemäß den Anforderungen 1 bis 4 von RFC 6960, Abschnitt 3.2, empfangen und akzeptiert wird, muss festgestellt werden, dass das Zertifikat des Unterzeichners zum Zeitpunkt der Unterzeichnung nicht widerrufen war, wenn eine der folgenden Anforderungen erfüllt ist:

- Der Anspruch signature hat eine gültigen Zeitstempel, und die beglaubigten Zeit fällt innerhalb das `(thisUpdate,nextUpdate)` der Antwort, oder
- Die Anspruchsignatur hat keinen gültigen Zeitstempel, aber die aktuelle Echtzeit liegt innerhalb des `(thisUpdate,nextUpdate)`-Intervall der Antwort, und beide der

folgenden Anforderungen sind erfüllt:

- Das Feld `„certStatus"` der Antwort ist `„gut"` oder `„widerrufen"`, aber mit dem `Widerrufsgrund removeFromCRL`, und
- Der OCSP-Signierer der Antwort ist ein „autorisierter Responder" gemäß der Definition in RFC 6960, Abschnitt 4.2.2.2.

Wenn das Feld `„certStatus"` der Antwort `den Status „revoked" (widerrufen)` hat, aber mit einem `Widerrufsgrund`, der nicht `„removeFromCRL"` ist, muss festgestellt werden, dass das Zertifikat des Unterzeichners zum Zeitpunkt der Unterzeichnung nicht widerrufen war, wenn beide der folgenden Anforderungen erfüllt sind:

- Das Manifest hat einen gültigen Zeitstempel, und die beglaubigte Zeit liegt innerhalb des Intervalls `(thisUpdate,nextUpdate)` der Antwort und
- Die `revocationTime` in der Antwort liegt nach dem beglaubigten Zeitstempel.

Wenn die oben genannten Bedingungen erfüllt sind, gilt das Zertifikat zum Zeitpunkt der Signatur als nicht widerrufen, und der Validierer gibt den Erfolgscode `„signingCredential.ocsp.notRevoked"` aus.

Andernfalls:

- Wenn das certStatus-Feld von die Antwort ist `unbekannt`, der Anspruch soll abgelehnt werden mit einem `signingCredential.ocsp.unknown` Fehlercode.
- Andernfalls gilt das Zertifikat zum Zeitpunkt der Signatur als widerrufen und der Anspruch wird mit dem Fehlercode Fehlercode `„signingCredential.ocsp.revoked"` abgelehnt.

15.10. Validieren Sie die Assertions

15.10.1. Überprüfen Sie die korrekten Aussagen für den Manifesttyp

15.10.1.1. Allgemeines

Je nach Manifesttyp gibt es Aussagen, die entweder erforderlich oder verboten sind. Ein Validierer muss auf erforderliche und nicht zulässige Aussagen prüfen.

15.10.1.2. Standard-Manifest-Assertionen

Wenn es sich um ein Standardmanifest handelt:

1. Überprüfen Sie, ob genau eine harte Bindung an die Inhaltsaussage vorhanden ist – entweder eine `c2pa.hash.data`, eine `c2pa.hash.boxes`, eine `c2pa.hash.collection.data`, eine `c2pa.hash.bmff.v2` (veraltet) oder eine `c2pa.hash.bmff.v3`. Wenn keine solche Assertion vorhanden ist, muss das Manifest mit dem Fehlercode `claim.hardBindings.missing` abgelehnt werden. Wenn mehr als eine solche Assertion vorhanden ist, muss das Manifest mit d e m Fehlercode `assertion.multipleHardBindings` abgelehnt werden.
2. Überprüfen Sie, ob es null oder eine `c2pa.`ingredient-Assertion gibt, deren `Beziehung parentOf` ist. Wenn es mehr als eine gibt, wird das Manifest mit dem Fehlercode `manifest.multipleParents` abgelehnt.
3. Überprüfen Sie, ob entweder eine `c2pa.created-` oder eine `c2pa.`opened-Aktion in genau einer actions-Assertion enthalten ist.

15.10.1.3. Manifest-Assertions aktualisieren

Wenn es sich um ein Aktualisierungsmanifest handelt:

1. Überprüfen Sie, ob genau eine Inhaltsstoffangabe vorhanden ist und ob deren `Beziehung „parentOf“` lautet. Ist dies nicht der Fall (d. h. wenn sie fehlt, wenn mehr als eine vorhanden ist oder wenn der Wert `der Beziehung` nicht `„parentOf“` lautet), wird das Manifest mit dem Fehlercode `„manifest.update.wrongParents“` abgelehnt.
2. Überprüfen Sie, ob keine `c2pa.hash.data-`, `c2pa.hash.boxes-`, `c2pa.hash.collection.data-`, `c2pa.hash.bmff.v2-` (veraltet), `c2pa.hash.bmff.v3-` oder Thumbnail-Assertions vorhanden sind. Ist dies der Fall, wird das Manifest mit dem Fehlercode `manifest.update.invalid` abgelehnt.
3. Überprüfen Sie, dass keine `c2pa.hash.multi-asset`-Assertions vorhanden sind. Ist dies der Fall, wird das Manifest mit dem Fehlercode `manifest.update.invalid` abgelehnt.
4. Wenn eine oder mehrere `c2pa.actions`- oder `c2pa.actions.`v2-Assertions vorhanden sind, überprüfen Sie, ob das Aktionsfeld jeder Aktion, die im Aktionsarray einer solchen Assertion gefunden wurde, einer der unterstützten Werte ist, die unter „Manifeste aktualisieren“ angegeben sind. Ist dies nicht der Fall, wird das Manifest mit dem Fehlercode `manifest.update.invalid` abgelehnt.

15.10.2. Vorbereitung der Liste der redigierten Assertions

Bei der Verarbeitung eines Anspruchs muss ein Validierer die Gruppe der redigierten Assertions für das Manifest jedes Inhaltsstoffs (sofern vorhanden) auf der Grundlage jeder in seinem Feld `„redacted_assertions“` aufgeführten JUMBF-URI zusammenstellen. Das Feld `„redacted_assertions“` eines Anspruchs darf niemals eine JUMBF-URI zu einer seiner eigenen Assertions enthalten.

HINWEIS	Behauptungen können zu jedem Zeitpunkt der Herkunft des endgültigen Assets aus den Inhaltsstoffen entfernt werden. und nicht unbedingt durch den Anspruchgenerator, der eine Inhaltsstoffdatei als Inhaltsstoff zuerst verwendet.

Weitere Informationen finden Sie im Abschnitt 15.11.3.2, „Durchführung einer expliziten Validierung“.

15.10.3. Validierung von Behauptungen

15.10.3.1. Allgemeines

Jede Assertion in den Feldern „created_assertions“ und „gathered_assertions“ des Anspruchs (und im Feld „assertions“ eines v1-Anspruchs) ist eine „hashed_uri“-Struktur. Für jede Assertion muss der Validator zunächst feststellen, ob die URI-Referenz im Feld „url“ in der Liste der redigierten Assertionen enthalten ist.

NOTE Auch wenn die im Feld „gathered_assertions” aufgeführten Behauptungen nicht vom , sind sie dennoch Teil des Anspruchs und werden daher ebenfalls gemäß diesem Validierungsalgorithmus validiert.

Befindet sich die Behauptung in der Liste der redigierten Behauptungen und lautet ihre Bezeichnung „c2pa.actions“ oder „c2pa.actions.v2“, wird der Anspruch mit dem Fehlercode „assertion.action.redacted“ abgelehnt, da Behauptungen vom Typ „c2pa.actions“ und „c2pa.actions.v2“ nicht redigiert werden dürfen. Wenn sie in der Liste der redigierten Assertions enthalten ist und die Bezeichnung der Assertion eine feste Bindung an eine Inhaltsassertion ist – entweder c2pa.hash.data, c2pa.hash.boxes, c2pa.hash.collection.data, c2pa.hash.bmff.v2 (veraltet) oder c2pa.hash.bmff.v3 –, wird der Anspruch mit dem Fehlercode assertion.dataHash.redacted abgelehnt, da diese Arten von Assertions nicht redigiert werden dürfen. Andernfalls wird die redigierte Assertion als gültig angesehen und die Validierung wird entsprechend der Art der Assertion fortgesetzt.

Für alle anderen Behauptungen (die nicht in der Liste der redigierten Behauptungen enthalten sind) wird die URI-Referenz im URL-Feld aufgelöst, um die Daten abzurufen. Wenn die URI nicht auf einen Ort innerhalb desselben C2PA-Manifests verweist (ein self#jumbf-Ort), wird die Behauptung mit dem Fehlercode assertion.outsideManifest abgelehnt. Wenn die URI nicht aufgelöst und die Daten nicht abgerufen werden können, wird die Behauptung mit dem Fehlercode assertion.missing abgelehnt.

Befolgen Sie das Verfahren in Abschnitt 15.4, „Bestimmung des Hash-Algorithmus“, um den Hash-Algorithmus und mögliche Fehlercodes zu bestimmen. Berechnen Sie den Hashwert der Assertion unter Verwendung dieses Algorithmus und des in Abschnitt 8.4.2.3, „Hashing von JUMBF-Boxen“, beschriebenen Verfahrens und vergleichen Sie den berechneten Hashwert mit dem Wert im Hashfeld. Wenn sie nicht übereinstimmen, wird der Anspruch mit dem Fehlercode „assertion.hashedURI.mismatch“ abgelehnt. Andernfalls wird der Erfolgscode „assertion.hashedURI.match“ aufgezeichnet.

Wenn der Inhalt einer Standardaussage kein wohlgeformtes CBOR oder kein konformes JSON ist, wird der Anspruch mit dem Fehlercode assertion.cbor.invalid oder assertion.json.invalid abgelehnt.

HINWEIS Wohlgeformtes CBOR ist in RFC 8949, Anhang C, definiert.

HINWEIS RFC 8259, Abschnitt 2, definiert die Grammatik, der JSON-Daten entsprechen müssen.

Wenn eine im Assertion Store vorhandene Assertion nicht durch ein Element der Arrays created_assertions oder gathered_assertions im Anspruch (oder dem Array assertions im v1-Anspruch) referenziert wird, wird der Anspruch mit dem Fehlercode assertion.undeclared abgelehnt.

Für jede URI im Array „redacted_assertions” des Anspruchs gilt: Wenn die URI auf das eigene Manifest des Anspruchs verweist, wird der Anspruch mit dem Fehlercode „assertion.selfRedacted” abgelehnt. Ein Anspruch darf seine eigenen

Aussagen redigieren.

15.10.3.2. Spezifische Assertion-Validierung

Für jede Behauptung überprüft der Validierer die Bezeichnung der Behauptung. Ist diese unten aufgeführt, führt der Validierer die spezifischen Validierungsschritte für diesen Behauptungstyp durch. Ist die Bezeichnung der Behauptung unten nicht aufgeführt, sind für diesen Behauptungstyp keine zusätzlichen Validierungsschritte erforderlich, die über die bereits beschriebenen hinausgehen.

- `c2pa.cloud-data`, Abschnitt 15.10.3.2.1, „`c2pa.cloud-data`-Validierung"
- `c2pa.actions` oder `c2pa.actions.v2`, Abschnitt 15.10.3.2.2, „`c2pa.actions`-Validierung"
- `c2pa.metadata`, Abschnitt 15.10.3.2.3, „`c2pa.metadata`-Validierung"

HINWEIS	Zutat Aussagen (`c2pa.ingredient` oder `c2pa.Zutat.v2` oder `c2pa.ingredient.v3`) unterliegen einer zusätzlichen Validierung an einer anderen Stelle im Validierungsprozess (siehe Abschnitt 15.11, „Validierung der Inhaltsstoffe").

Wenn der Wert eines Feldes einer Standardaussage ein `hashed_uri` oder `hashed_ext_uri` ist, muss der Validierer die in Abschnitt 15.10.3.3, „Validierung von Referenzen", beschriebenen Schritte ausführen, mit Ausnahme des Feldes `activeManifest` in `c2pa.ingredient.v3`, für das in Abschnitt 15.11.3, „Validierung von Inhaltsstoffangaben" festgelegt ist.

15.10.3.2.1. `c2pa.cloud-data`-Validierung

Wenn die Bezeichnung der Assertion `c2pa.cloud-data` lautet:

1. Überprüfen Sie, ob die Assertion die folgenden Felder enthält: `label`, `size`, `location` und `content_type`. Wenn eines dieser Felder fehlt, muss die Assertion mit dem Fehlercode `assertion.cloud-data.malformed` abgelehnt werden.
2. Wenn das Label-Feld der externen Assertion `c2pa.hash.data`, `c2pa.hash.boxes`, `c2pa.hash.collection.data`, `c2pa.hash.bmff.v2` (veraltet) oder `c2pa.hash.bmff.v3` lautet, wird die Assertion mit dem Fehlercode `assertion.cloud-data.hardBinding` abgelehnt.
3. Wenn es sich bei dem Manifest um ein Aktualisierungsmanifest handelt und das Label-Feld der externen Assertion `c2pa.actions` oder `c2pa.actions.v2`, wird der Anspruch mit einem Fehlercode von `assertion.cloud-data.actions` abgelehnt.
4. Das Feld „`location`" muss gemäß Abschnitt 15.10.4.2, „Validierung externer Referenzen", validiert werden.

15.10.3.2.2. `c2pa`.actions-Validierung

Wenn die Bezeichnung der Assertion `c2pa.actions` oder `c2pa.actions.v2` lautet:

1. Stellen Sie sicher, dass ein Feld „`actions`" enthält. Wenn nicht, der Anspruch abgelehnt mit einem Fehlercode von `assertion.action.malformed`.
2. `Für` jede Aktion in der Aktionsliste:
   a. Wenn das Aktionsfeld entweder `c2pa.created` oder `c2pa.opened` ist, wird der Anspruch mit einem

Fehlercode von `assertion.action.malformed`, es sei denn, alle folgenden Bedingungen sind erfüllt:

i. Die Assertion ist die erste Aktionsassertion im Array `created_assertions` oder `gathered_assertions` (eines v2-Claims) oder die erste actions-Assertion im assertions-Array eines v1-Claims ist und

ii. die Aktion ist das erste Element im Array „`actions`" in dieser Assertion.

b. Wenn das Aktionsfeld `c2pa.opened`, `c2pa.placed` oder `c2pa.removed` lautet:

i. Wenn die Aktion kein Parameterfeld hat oder der Wert dieses Feldes leer ist, wird der Anspruch mit dem Fehlercode `assertion.action.ingredientMismatch` abgelehnt.

ii. Wenn das Feld „action's `parameters`" kein Feld „`ingredients`" (oder „`ingredient`" für `c2pa.actions`) enthält, wird der Antrag mit dem Fehlercode `assertion.action.ingredientMismatch` abgelehnt.

iii. Wenn der Wert des Feldes „`Zutaten`" kein Array mit mindestens einem Element ist, wird der Antrag mit dem Fehlercode `assertion.action.ingredientMismatch` abgelehnt.

iv. Überprüfen Sie Verweise auf Zutatenangaben:

A. Für `c2pa.opened`: Überprüfen Sie, ob das Feld „`Zutaten`" (oder das Feld „`Zutaten`" für `c2pa.actions`) genau eine gültige Hash-URI enthält, die zu einer Zutatenaussage im aktuellen Manifest aufgelöst werden kann, deren Beziehungsfeld „`parentOf`" ist. Ist dies nicht der Fall, wird der Anspruch mit dem Fehlercode „`assertion.action.ingredientMismatch`" abgelehnt.

B. Für `c2pa.placed`: Überprüfen Sie, ob das Feld „`ingredients`" (oder das Feld „`ingredient`" für `c2pa.actions`) eine oder mehrere gültige Hash-URIs enthält, die jeweils zu einer Zutatenangabe im aktuellen Manifest aufgelöst werden können, deren Beziehungsfeld „`componentOf`" lautet. Ist dies nicht der Fall, wird der Anspruch mit dem Fehlercode „`assertion.action.ingredientMismatch`" abgelehnt.

C. Für `c2pa.removed`: Überprüfen Sie, ob das Feld „`ingredients`" (oder das Feld „`ingredient`" für `c2pa.actions`) eine oder mehrere gültige Hash-URIs enthält, die jeweils zu einer Zutatenangabe in einem anderen Manifest aufgelöst werden können, dessen Beziehungsfeld „`componentOf`" lautet. Ist dies nicht der Fall, wird die Angabe mit dem Fehlercode „`assertion.action.ingredientMismatch`" abgelehnt.

c. Wenn das Feld „`action`" `c2pa.transcoded` oder `c2pa.repackaged` lautet:

i. Wenn das Feld „`ingredients`" (oder das Feld „`ingredient`" für `c2pa.actions`) vorhanden ist, überprüfen Sie, ob jedes Element dieses Feldes eine gültige Hash-URI ist, die zu einer Zutatenangabe im aktuellen Manifest mit der Beziehung „`parentOf`" aufgelöst werden kann. Ist dies nicht der Fall, wird der Anspruch mit dem Fehlercode „`assertion.action.ingredientMismatch`" abgelehnt.

d. Wenn das Aktionsfeld `c2pa.redacted` lautet:

i. Überprüfen Sie das `redigierte` Feld, das Teil des Parameterobjekts ist, auf das Vorhandensein einer JUMBF-URI. Wenn die JUMBF-URI nicht vorhanden ist oder nicht zu einer Assertion aufgelöst werden kann, wird der Anspruch mit dem Fehlercode `assertion.action.redactionMismatch` abgelehnt.

e. Wenn es ein Feld „`softwareAgent`" in der `action-common-map-v2` oder einen oder mehrere `softwareAgents` im Feld `softwareAgents` der Datei `actions-map-v2` aufgeführt sind:

i. Wenn in der `generator-info-map` ein Feld „`icon`“ vorhanden ist, muss es gemäß Abschnitt 15.10.3.3, „Validierung von Referenzen“, validiert werden.

f. Für jede Vorlage in der Vorlagenliste:

i. Wenn in der `action-template-map-v2` ein Feld „`icon`“ vorhanden ist, muss es gemäß Abschnitt 15.10.3.3, „Validierung von Referenzen“, validiert werden.

15.10.3.2.3. `c2pa.metadata`-Validierung

Wenn die Bezeichnung der Assertion „`c2pa.metadata`“ lautet, muss der Validator sicherstellen, dass die Assertion keine Felder enthält, die nicht in der zulässigen Liste aufgeführt sind. Wenn ein in der Assertion enthaltenes Feld nicht in der zulässigen Liste aufgeführt ist, muss der Anspruch mit dem Fehlercode „`assertion.metadata.disallowed`“ abgelehnt werden.

HINWEIS Diese Validierungsanforderung erfordert, dass ein Validator die in der Assertion enthaltenen JSON-LD-Daten parst.

15.10.3.2.4. `c2pa.time-stamp`-Validierung

Wenn die Bezeichnung der Assertion „`c2pa.time-stamp`“ lautet, muss der Validator sicherstellen, dass die Assertion ein wohlgeformtes CBOR ist, das aus einer einzigen Map (Major-Typ `5`) mit mindestens einem Schlüssel/Wert-Paar besteht. Ist dies nicht der Fall, wird der Anspruch mit dem Fehlercode „`assertion.timestamp.malformed`“ abgelehnt.

Da die Validierung des Zeitstempel-Tokens wie in Abschnitt 15.8.2, „Validierung des TimeStampToken“, beschrieben durchgeführt wird, muss der Validator das Zeitstempel-Token (und die zugehörige C2PA-Manifest-Kennung) für die spätere Verwendung speichern.

15.10.3.3. Validierung von Referenzen

Einige C2PA-Standardaussagen unterstützen die Verweisung auf andere Felder im C2PA-Manifest durch die Verwendung von `hashed_uri` und `hashed_ext_uri`. Beispielsweise kann es verschiedene Verweise in Aussagen zu Aktionen, Inhaltsstoffen und Miniaturansichten geben.

Für alle `hashed_uri`- und hashed_ext_uri-Felder in Standardaussagen, mit Ausnahme des Feldes `activeManifest` in `c2pa.ingredient.v3` (für das in Abschnitt 15.11.3, „Validierung von Inhaltsstoffaussagen“, ein spezielles Validierungsverhalten festgelegt ist), muss der Validator die folgende Validierung durchführen: . Für ein `hashed_ext_uri`, dessen Ressource der Validator abrufen möchte, muss der Validator die in Abschnitt 15.10.4.2, „Validierung externer Verweise“, beschriebenen Schritte durchführen. Für ein `hashed_uri` muss der Validator die unten beschriebenen Schritte durchführen.

Das Ziel eines `hashed_uri` befindet sich in seinem url-Feld. Ist das Feld nicht vorhanden oder kann das Ziel nicht gefunden werden (d. h. die Daten befinden sich nicht an der vorgesehenen Stelle), ist dies als Validierungsfehler mit dem Code `hashedURI.missing` zu behandeln.

Wenn das Ziel gefunden werden kann, gehen Sie wie folgt vor: . Befolgen Sie die Vorgehensweise in Abschnitt 15.4, „Bestimmen des Hash-Algorithmus“, um den Hash-Algorithmus und mögliche Fehlercodes zu bestimmen. . Stellen Sie sicher, dass das Hash-Feld in der Struktur „`hashed_uri`“ vorhanden ist. Ist dies nicht der Fall, wird die Anforderung mit dem Fehlercode „`hashedURI.mismatch`“ abgelehnt. . Berechnen Sie den Hash der Assertion unter Verwendung des bestimmten Hash-Algorithmus und des

in Abschnitt 8.4.2.3, „Hashing von JUMBF-Boxen“, beschriebenen Verfahren. . Vergleichen Sie den berechneten Hash-Wert mit dem Wert im Hash-Feld. Wenn sie nicht übereinstimmen, wird der Anspruch mit dem Fehlercode `hashedURI.mismatch` abgelehnt.

15.10.4. Validierung externer Daten

15.10.4.1. Allgemeines

Der Inhalt einer Cloud-Datenaussage enthält die URI-Referenzen und Hashes externer Daten und wird wie jede andere Aussage validiert, aber diese Referenzen werden nicht als Teil der Standardvalidierung abgerufen und validiert. Ein Validierer muss zunächst eine Aussage erfolgreich validieren, bevor er versucht, die referenzierten externen Daten abzurufen. Ein Validierer darf nicht versuchen, externe Daten aus einer abgelehnten Behauptung abzurufen. Da das Abrufen externer Daten optional ist, führt die Unmöglichkeit, externe Daten abzurufen oder zu validieren, nicht dazu, dass eine Behauptung abgelehnt wird.

Wenn ein Validierer beschließt, externe Daten in einer Cloud-Datenaussage abzurufen, muss er die in Abschnitt 15.10.4.2, „Validierung externer Referenzen“, beschriebenen Schritte ausführen.

15.10.4.2. Validierung externer Referenzen

Das folgende Verfahren ist zur Validierung der in einer Cloud-Datenaussage referenzierten externen Daten anzuwenden:

1. Lösen Sie die URI-Referenz im Feld `„url“` auf, um die Daten abzurufen. Wenn das Feld `„url“` nicht vorhanden ist oder die URI nicht aufgelöst und die Daten nicht abgerufen werden können, muss der Validator den Versuch, die externen Daten abzurufen, abbrechen.
2. Wenn die Größe der abgerufenen Daten nicht mit dem Wert des Feldes `„size“` übereinstimmt, gibt der Validator einen Fehlercode `„assertion.hashedURI.mismatch“` an die Anwendung zurück und stellt die abgerufenen Daten nicht bereit.
3. Überprüfen Sie, ob der im Content-Type-Header der HTTP-Antwort zurückgegebene Inhaltstyp mit dem deklarierten Inhaltstyp übereinstimmt. Wenn sie nicht übereinstimmen, gibt der Validator einen Fehlercode von `assertion.hashedURI.mismatch` an die Anwendung zurück und stellt die abgerufenen Daten nicht bereit. Der deklarierte Inhaltstyp wird bestimmt durch:
    a. Bei externen Daten wird der Inhaltstyp durch das Feld `dc:format` der Struktur `hashed_ext_uri`. Wenn das Feld `„dc:format“` fehlt, ist die Validierung des Inhaltstyps immer erfolgreich.
    b. Bei einer Cloud-Datenaussage bestimmt das Feld `„dc:format“`, sofern es im Feld `„location“` vorhanden ist, den Inhaltstyp, und der Wert des Feldes `„content_type“` der Cloud-Datenaussage wird ignoriert. Wenn `„location“` kein Feld `„dc:format“` enthält, bestimmt das Feld `„content_type“` der Aussage den Inhaltstyp.
4. Bestimmen Sie den zu verwendenden Hash-Algorithmus gemäß Abschnitt 15.4.2, „Für gehashtete Ext-URIs“ oder mögliche Fehlercodes.
5. Berechnen Sie den Hash der Daten unter Verwendung des festgelegten Hash-Algorithmus und des in Abschnitt 8.4.2.3, „Hashing von JUMBF-Boxen“, beschriebenen Verfahrens für den abgerufenen Inhalt. Verwenden Sie für externe Daten den Hash-Algorithmus und den exakt abgerufenen Inhalt als Eingabe für die Hash-Funktion.
    a. Vergleichen Sie den berechneten Hash-Wert mit dem Wert im Hash-Feld. Wenn das Hash-Feld nicht vorhanden ist oder die Werte nicht übereinstimmen, muss der Validator einen Fehlercode von `assertion.hashedURI.mismatch` an die Anwendung zurückgeben und darf die abgerufenen Daten nicht bereitstellen.

b. Andernfalls muss der Validator einen Erfolgscode von `assertion.hashedURI.match` aufzeichnen und die abgerufenen Daten an die Anwendung übermitteln.

15.11. Überprüfen Sie die Inhaltsstoffe

15.11.1. Erläuterung

Ein Validierer muss die Validierungsschritte für das vorgelegte Asset und dessen aktives Manifest durchführen. Wenn einer der Schritte zu dem Ergebnis führt, dass das aktive Manifest ungültig ist, muss dieses Manifest mit dem angegebenen Fehlercode abgelehnt werden.

Das aktive Manifest eines Vermögenswerts kann durch die Verwendung von Inhaltsstoffangaben einen oder mehrere Inhaltsstoffe aufführen. Einige dieser Inhaltsstoffe können mit eigenen Manifesten verknüpft sein, und einige dieser Manifeste können wiederum selbst Inhaltsstoffe und Inhaltsstoffmanifeste enthalten.

15.11.2. Verarbeitung von Inhaltsstoffmanifesten

15.11.2.1. Standardmanifeste in einem Bestandteil

Bei der Verarbeitung eines Standardmanifests muss ein Validierer jede Zutat validieren (unabhängig vom Wert ihres Beziehungsfeldes) wie unten beschrieben validieren.

15.11.2.2. Aktualisieren von Manifesten in einer Zutat

Bei Aktualisierungsmanifesten muss die „parentOf“-Zutat des Aktualisierungsmanifests wie unten beschrieben validiert werden.

15.11.2.3. Zeitstempel-Manifeste in einer Komponente

WICHTIG	Diese Funktion wurde zugunsten der Zeitstempel-Assertion als veraltet deklariert. Die folgenden Informationen werden aus historischen Gründen beibehalten.

Alle Zeitstempel-Manifeste, die in einer Komponente gefunden werden, sind zu ignorieren.

15.11.3. Validierung der Assertion einer Komponente

15.11.3.1. Übersicht über die Validierung

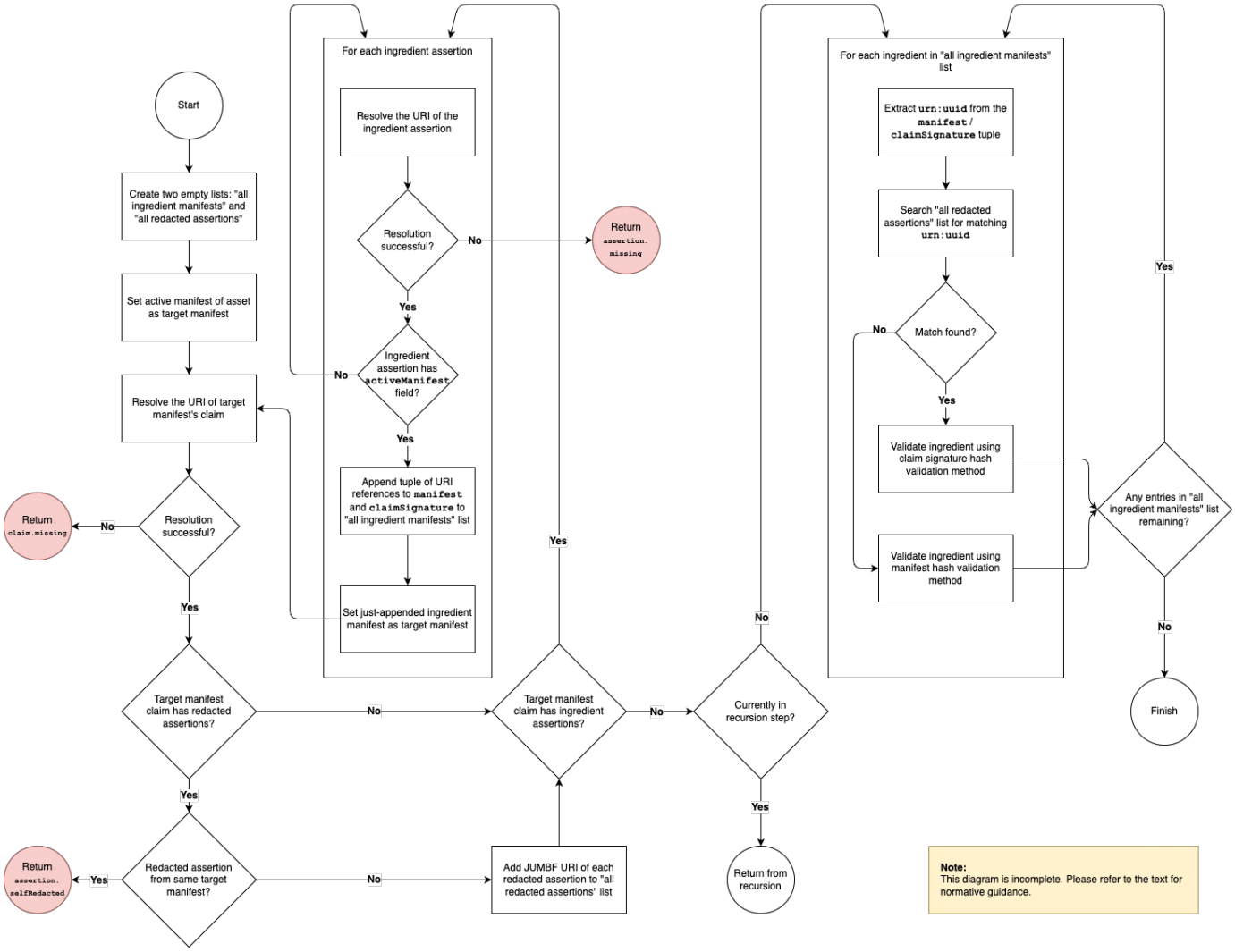

Abbildung 14. Validierung von Inhaltsstoffen

Das Flussdiagramm in Abbildung 14, „Validierung von Inhaltsstoffen", beschreibt den Prozess der Validierung aller Inhaltsstoffangaben, die in einem bestimmten C2PA-Manifest enthalten sind.

HINWEIS

Bei Abweichungen zwischen der visuellen Darstellung und dem Text ist der Text maßgebend.

15.11.3.2. Durchführung einer expliziten Validierung

Wenn das Beziehungsfeld in einer Zutatenangabe nicht vorhanden ist, wird die Angabe mit dem Fehlercode „`assertion.ingredient.malformed`" abgelehnt.

Der Wert des Beziehungsfeldes muss einer der folgenden sein: `parentOf`, `inputTo` oder `componentOf`. Wenn der Wert des Beziehungsfeldes keiner dieser Werte ist, wird die Angabe mit dem Fehlercode `assertion.ingredient.malformed` abgelehnt.

15.11.3.3. Durchführung einer rekursiven Validierung

Der Validator muss alle Inhaltsstoffmanifeste im Asset rekursiv validieren, beispielsweise mithilfe einer Tiefensuche wie

wie unten beschrieben. Ein Validierer muss den Algorithmus nicht genau wie beschrieben implementieren, aber die Ergebnisse der Validierung müssen den Ergebnissen dieses Algorithmus entsprechen.

1. Erstellen Sie zwei leere Listen:
    a. Eine Liste, die die `hashed_uri`-Werte aller in der Ressource verwendeten Inhaltsverzeichnisse enthält, unabhängig davon, wo sie in der Herkunftskette vorkommen.
    b. Eine Liste zur Speicherung der JUMBF-URIs aller redigierten Assertions in der Ressource, unabhängig davon, wo sie in ihrer Herkunftskette vorkommen.
2. Legen Sie das aktive Manifest des zu validierenden Assets als Zielmanifest fest.
3. Rekursion starten.
4. Suchen Sie den Anspruch, wie in Abschnitt 15.6, „Suchen und Validieren des Anspruchs“, beschrieben. Ist dies nicht möglich, lehnen Sie den Anspruch mit dem Fehlercode `claim.missing` ab.
5. Wenn der Anspruch des Zielmanifests ein Feld `„redacted_assertions“` enthält, überprüfen Sie die JUMBF-URI jeder redigierten Behauptung.
    a. Wenn die redigierte Assertion aus dem Zielmanifest stammt, lehnen Sie den Anspruch mit dem `Fehlercode assertion.selfRedacted`
    Fehlercode zurück.
    b. Andernfalls fügen Sie die JUMBF-URI der redigierten Aussage an die Liste aller redigierten Aussagen an.
6. Wenn die Angabe des Zielmanifests Inhaltsstoffangaben enthält:
    a. Für jede Inhaltsstoffaussage:
        i. Versuchen Sie, die gehashte URI der Inhaltsstoffaussage aufzulösen. Wenn die URI nicht aufgelöst werden kann, der Hash nicht übereinstimmt oder die JUMBF-Inhaltsfelder der Aussage nur Nullen enthalten, fahren Sie mit der nächsten Inhaltsstoffaussage fort.
        ii. Wenn die Inhaltsstoffaussage ein Feld `„activeManifest”` (oder ein Feld `„c2pa_manifest”` in einer Inhaltsstoffaussage der Version 1 oder 2) enthält:
            A. Fügen Sie der Liste aller Inhaltsstoffmanifeste ein Tupel hinzu, das die folgenden Werte enthält:
                - Der Wert `„hashed_uri”` des Feldes `„activeManifest”` (oder `„c2pa_manifest”`) in der Zutatenangabe
                - Der Wert `„hashed_uri”` des Feldes `„claimSignature”` in der Zutatenangabe
            B. Legen Sie das gerade angehängte Ingredient Manifest als Zielmanifest fest und wiederholen Sie den oben beschriebenen Vorgang ab dem Schritt „Beginn der Rekursion”.
        iii. Wenn die Inhaltsangabe kein Feld `„activeManifest“` (oder `„c2pa_manifest“`) enthält, einen Informationscode `„ingredient.unknownProvenance“` aufzeichnen, es sei denn, der Wert des Feldes `„relationship“` ist `„inputTo“`, und dann zur nächsten Inhaltsangabe springen, bis alle erschöpft sind. An diesem Punkt von der aktuellen Rekursionsstufe zurückkehren.
7. Wenn die Angabe des Zielmanifests keine Inhaltsstoffangaben enthält, kehren Sie von der aktuellen Rekursionsstufe zurück.
8. Beenden Sie die Rekursion.

Nachdem der Validierer eine Liste aller Inhaltsstoffangaben und eine Liste aller redigierten Aussagen zusammengestellt hat, führt er den folgenden Validierungsalgorithmus durch

folgenden Validierungsalgorithmus durch:

1. Für jedes Inhaltsstoffverzeichnis in der Liste aller Inhaltsstoffverzeichnisse:
    a. Extrahieren Sie die Manifest-Kennzeichnung aus dem Inhaltsstoff-Manifest JUMBF-URI aus jedem Tupel
    b. Durchsuche die Liste aller redigierten Behauptungen nach Behauptungen mit einem übereinstimmenden Manifest-Label
    c. Wenn eine oder mehrere übereinstimmende redigierte Behauptungen gefunden werden:
        i. Überprüfen Sie die Zutat mithilfe der in Abschnitt 15.11.3.3.1, „Überprüfungsmethode für die Signatur der Angabe“, beschriebenen Überprüfungsmethode für die Signatur der Angabe.
    d. Wenn keine übereinstimmenden redigierten Assertions gefunden werden:
        i. Validieren Sie die Zutat entweder mit der Manifest-Hash-Validierungsmethode, die in Abschnitt 15.11.3.3.2, „Manifest-Hash-Validierungsmethode“, beschrieben ist, oder mit der Anspruchsignatur-Hash-Validierungsmethode, die in Abschnitt 15.11.3.3.1, „Anspruchsignatur-Hash-Validierungsmethode“, beschrieben ist.
    e. Wenn die Inhaltsstoffaussage ein Feld „`validationsResults`“ enthält:
        i. Geben Sie für jeden Eintrag im Wert des Feldes „`validationsResults`“ einen entsprechenden Eintrag als Teil der Validierungsergebnisse zurück, sofern dieser nicht bereits im Rahmen des Validierungsprozesses zurückgegeben wurde.
        ii. Wenn im Rahmen des Validierungsprozesses Einträge zurückgegeben werden, die nicht im Feld Feld „`validationResults`“ `vorhanden sind`, geben Sie diese als Teil der Validierungsergebnisse zurück.
    f. Wenn kein Feld „`validationResults`“ vorhanden ist und es sich bei der Zutatenangabe um eine v3-Zutatenangabe mit dem Feld Feld „`activeManifest`“ vorhanden ist, dann den Fehlercode „`assertion.ingredient.malformed`“ zurück.

Validatoren sollten alle zusätzlichen C2PA-Manifeste ignorieren, die im C2PA-Manifest-Speicher erscheinen, aber nicht in der Liste der Inhaltsstoffmanifeste enthalten sind.

HINWEIS	Das Ignorieren zusätzlicher C2PA-Manifeste unterstützt die Kompatibilität mit benutzerdefinierten Assertions und zukünftigen Konstrukten, die möglicherweise auf C2PA-Manifeste in einer Weise verweisen, die der Validator nicht erkennt.

15.11.3.3.1. Methode zur Validierung der Signatur-Hash-Anforderung

Diese Methode umfasst eine vollständige Validierung der Angabe der Zutat, wie sie für das aktive Manifest durchgeführt wird, mit der Ausnahme, dass Inhaltsbindungen nicht ausgewertet werden:

1. Lösen Sie die URI-Referenz im URL-Wert des Feldes „`claimSignature`“ auf, um das Feld „Claim Signature“ der Zutat zu erhalten. Wenn die URI-Referenz nicht aufgelöst werden kann oder das Feld „`claimSignature`“ nicht vorhanden ist, wird die Zutatenangabe mit dem Fehlercode „`ingredient.claimSignature.missing`“ abgelehnt.
2. Bestimmen Sie die Hash-Algorithmus-Kennung (oder den möglichen Fehlercode), indem Sie das Verfahren in Abschnitt 15.4, „Bestimmen des Hash-Algorithmus“, befolgen.
3. Berechnen Sie den Hashwert des Feldes „Ingredient Claim Signature“ (Angabe zu Inhaltsstoffen) unter Verwendung dieses Algorithmus und des in Abschnitt 8.4.2.3, „Hashing JUMBF Boxes“ (Hashwertberechnung für JUMBF-Felder), beschriebenen Verfahrens.
4. Vergleichen Sie den berechneten Hashwert mit dem Wert im Hashfeld.

a. Wenn die Hash-Werte nicht übereinstimmen oder das Hash-Feld nicht vorhanden ist:

   i. Lehnen Sie die Angabe mit dem Fehlercode `ingredient.claimSignature.mismatch` ab.

b. Wenn die Hash-Werte übereinstimmen, geben Sie einen Fehlercode „`ingredient.claimSignature.validated`“ aus.

   i. Validieren Sie die Signatur der Angabe, den Zeitstempel und die Informationen zur Sperrung der Berechtigungsnachweise gemäß Abschnitt 15.7, „Validieren der Signatur“, Abschnitt 15.8, „Validieren des Zeitstempels“, und Abschnitt 15.9, „Validieren der Informationen zur Sperrung der Berechtigungsnachweise“.

   ii. Für jede URI in der Liste der redigierten Assertions mit einer übereinstimmenden Manifest-Kennzeichnung gilt: Wenn die referenzierte Assertion vorhanden ist und ein JUMBF-Inhaltsfeld oder ein Auffüllfeld darin etwas anderes als null oder mehr 0x00-Bytes enthält, wird der Anspruch mit dem Fehlercode „`assertion.notRedacted`“ abgelehnt.

   iii. Validieren Sie jede nicht redigierte Behauptung gemäß Abschnitt 15.10, „Validieren der Behauptungen“, mit Ausnahme der fest gebundenen Behauptungen, die für Inhaltsstoffe nicht validiert werden können.

Bei Verwendung der Hash-Validierungsmethode für die Behauptungssignatur darf der Validator keine Hash-Fehlercodes für das Feld „`activeManifest`“ aufzeichnen.

ANMERKUNG Der Grund dafür ist, dass, wenn Redaktionen das referenzierte Manifest betreffen, es möglich ist, dass der Hash für dieses Feld nicht übereinstimmt.

15.11.3.3.2. Methode zur Validierung des Manifest-Hashwerts

Ein Manifest für Inhaltsstoffe, das aufgrund von Redaktionen nicht geändert wurde, kann schneller validiert werden, wenn der aktuelle Validator den Validierungsergebnissen des vorherigen Anspruchgenerators vertraut:

1. Lösen Sie die URI-Referenz im URL-Wert des Feldes „`activeManifest`“ auf, um das Manifestfeld der Zutat zu erhalten. Wenn das URL-Feld nicht vorhanden ist oder die URI-Referenz nicht aufgelöst werden kann, wird die Zutatenangabe mit dem Fehlercode „`ingredient.manifest.missing`“ abgelehnt.
2. Bestimmen Sie die Hash-Algorithmus-Kennung (oder den möglichen Fehlercode), indem Sie das Verfahren in Abschnitt 15.4, „Bestimmen des Hash-Algorithmus“, befolgen.
3. Berechnen Sie den Hashwert des Manifestfelds der Zutat unter Verwendung dieses Algorithmus und des in Abschnitt 8.4.2.3, „Hashwertberechnung für JUMBF-Felder“, beschriebenen Verfahrens.
4. Vergleichen Sie den berechneten Hash mit dem Wert im Hash-Feld.

   a. Wenn die Hash-Werte nicht übereinstimmen oder das Hash-Feld nicht vorhanden ist:

      i. Lehnen Sie den Anspruch mit dem Fehlercode `ingredient.manifest.mismatch` ab.

   b. Wenn die Hash-Werte ü bereinstimmen, ist die Zutat vollständig validiert und es wird ein Erfolgscode „`ingredient.manifest.validated`“ `ausgegeben.`
   ausgegeben.

15.12. Validieren Sie den Inhalt des Assets.

Der Inhalt des Assets wird anhand der festen Bindung im aktiven Manifest validiert, wenn es sich bei dem aktiven Manifest um ein Standardmanifest handelt. Handelt es sich bei dem aktiven Manifest um ein Aktualisierungsmanifest, muss die feste Bindung im Manifest der übergeordneten Komponente „`parentOf`“ gefunden werden

Manifest gefunden, oder, wenn dieses Manifest ebenfalls ein Aktualisierungsmanifest ist, indem der Kette von `parentOf`-Bestandteilen bis zum ersten Standardmanifest gefolgt wird. Wenn kein Standardmanifest gefunden wird oder das Standardmanifest keine feste Bindung enthält, wird die Angabe des aktiven Manifests mit dem Fehlercode `claim.hardBindings.missing` abgelehnt.

Ein Asset kann auch aus mehreren Teilen bestehen, wobei jeder Teil einen eigenen zugehörigen Hash hat (siehe Abschnitt 18.9, „Multi-Asset-Hash“), der separat validiert werden kann. Ein Asset kann beispielsweise aus separaten statischen Bild- und Videoteilen bestehen, die jeweils separat validiert werden können.

15.12.1. Validieren eines Daten-Hash

15.12.1.1. Allgemeines

Sobald ein Standardmanifest (und seine Bindungen) gefunden wurde, werden die Ausschlussbereiche aus der `c2pa.hash.`data-Assertion extrahiert.

Wenn der Endbyte-Offset eines Ausschlussbereichs (`Start` + `Länge`) größer ist als der Startbyte-Offset des nächsten Ausschlussbereichs im Array oder wenn ein `Start-` oder Längenwert negativ ist, wird das Manifest mit dem Fehlercode `assertion.dataHash.malformed` abgelehnt.

Wenn Aktualisierungsmanifeste gefunden wurden, wird der Längenwert des Ausschlussbereichs, dessen Startwert der Offset des Starts des gesamten C2PA-Manifestspeichers ist, als aktuelle Länge des gesamten C2PA-Manifestspeichers zuzüglich aller dateiformatspezifischen Extras behandelt.

Der in `c2pa.hash.data` angegebene Hash-Algorithmus (`alg`) wird über die Bytes des Assets berechnet, mit Ausnahme derjenigen, die im Ausschlussbereich angegeben sind. Wenn das Ende eines Ausschlussbereichs über das Ende des Assets hinausgeht, wird das Manifest mit dem Fehlercode `assertion.dataHash.mismatch` abgelehnt.

Wenn der im Feld „`alg`“ angegebene Hash-Algorithmus nicht in der Liste der zulässigen oder veralteten Algorithmen in Abschnitt 13.1, „Hashing“, aufgeführt ist, wird das Manifest mit dem Fehlercode „`algorithm.unsupported`“ abgelehnt. Wenn das Feld „`hash`“ nicht vorhanden ist, wird das Manifest mit dem Fehlercode „`assertion.dataHash.mismatch`“ abgelehnt.

Die Kombination aus Ausschlussbereichen und Auffüllwerten, insbesondere die Auffüllung, die zur Unterstützung von Mehrfachdurchlauf-Verarbeitungsworkflows erforderlich ist, kann es einem Angreifer ermöglichen, Teile dieser Auffüllung durch beliebige Daten zu ersetzen, die sich auf die Nutzung des Assets auswirken könnten, ohne den Hash ungültig zu machen. Aus diesem Grund muss ein Validierer sicherstellen, dass die im Ausschlussbereich enthaltenen Daten, einschließlich eines C2PA-Manifest-Speichers, nur aus dem C2PA-Manifest-Speicher und einer geeigneten Auffüllung (z. B. Null-Daten) in deutlich gekennzeichneten Auffüllfeldern oder Freiflächen/Überspringfeldern bestehen. In anderen Ausschlussbereichen als dem oben genannten C2PA-Manifest-Speicher können auch alle oder ein Teil der Asset-Metadaten enthalten sein, wie in Abschnitt 9.2.5, „Asset-Metadaten-Bindungen“, beschrieben. Wenn ein Validator auf andere als die oben zulässigen Daten stößt, muss das Manifest mit dem Fehlercode `assertion.dataHash.mismatch` abgelehnt werden. Wenn ein Validierer Ausschlussbereiche außerhalb des C2PA Manifest Store und entsprechende Auffüllungen (z. B. Null-Daten) in eindeutig gekennzeichneten Auffüllfeldern oder Freiflächen/Überspringfeldern findet, muss der Informationscode `assertion.dataHash.additionalExclusionsPresent` gesetzt werden.

Wenn keine Fehlerbedingungen aufgetreten sind, muss der Validator den Erfolgscode `assertion.dataHash.match` hinzufügen.
zur Liste, die schließlich zurückgegeben wird.

Wenn der über alle Daten des Vermögenswerts (abzüglich etwaiger Ausschlussbereiche) berechnete Hash nicht mit dem Wert des Hash-Feldes in `c2pa.hash.data` übereinstimmt, muss der Validator nach dem Vorhandensein einer Multi-Asset-Hash-Assertion suchen. Ist eine vorhanden, muss sie wie in Abschnitt 15.12.4, „Validieren eines Multi-Asset-Hash“, beschrieben validiert werden. Ist keine vorhanden, muss das Manifest mit dem Fehlercode `assertion.dataHash.mismatch` abgelehnt werden.

15.12.1.2. Hashing von JPEG 1-Dateien

In JPEG 1-Dateien würden die oben beschriebenen Dateiformat-Extras alle APP11-Marker und ihre jeweiligen Segmentlängenbytes für APP11-Segmente umfassen. Da die Segmentlängen innerhalb des Ausschlussbereichs liegen, muss ein Validator die Gesamtlänge des Ausschlussbereichs mit der Gesamtlänge aller APP11-Segmente, die das C2PA-Manifest darstellen, abgleichen, um sicherzustellen, dass die Länge nicht manipuliert wurde.

HINWEIS Eine JPEG 1-Datei kann aus anderen Gründen als C2PA (z. B. JPEG 360 oder JPEG Privacy and Security) APP11-Segmente enthalten, die in diesen Berechnungen nicht berücksichtigt werden.

15.12.2. Validierung eines BMFF-Hash

Für alle Teile eines Assets, die zur Darstellung für einen Benutzer gerendert werden, einschließlich, aber nicht beschränkt auf Audio, Video oder Text, muss die entsprechende feste Bindung, die dem gerenderten Inhalt entspricht, gemäß Abschnitt 9.2, „Feste Bindungen“, validiert werden. Wenn die standardmäßige feste Bindung nicht validiert wird und eine Multi-Asset-Hash-Assertion vorhanden ist, muss diese wie unter [validating_a_multi_asset_hash] beschrieben validiert werden. Wenn Inhalte zu irgendeinem Zeitpunkt nicht validiert werden können, muss der Validierer dem Benutzer deutlich signalisieren, dass ein Teil der Inhalte nicht mit der Angabe übereinstimmt, und wenn möglich angeben, welcher Teil der Inhalte nicht validiert wurde. Wenn Inhalte fehlen, für die Inhaltsbindungen existieren, gilt die Feststellung dieses Fehlens ebenfalls als Validierungsfehler. Der Validierer muss weiterhin melden, dass die Validierung fehlgeschlagen ist, auch wenn spätere Teile der Inhalte korrekt validiert werden.

Bei Inhalten, die vor Beginn der Wiedergabe nicht vollständig verfügbar sind, wie beispielsweise beim adaptiven Bitraten-Streaming (ABR) und beim progressiven Download, gilt das Fehlen noch nicht verfügbarer Teile des Inhalts nicht als Validierungsfehler. Sobald der Inhalt verfügbar ist, muss der Validator jeden Teil des Inhalts vor der Wiedergabe wie zuvor beschrieben validieren. Darüber hinaus muss der Validator überprüfen, ob die Reihenfolge der Inhalte mit der Reihenfolge übereinstimmt, die bei der Erstellung des Manifests festgelegt wurde. Sofern der Player dem Validator nicht ausdrücklich signalisiert hat, dass eine Unterbrechung zu erwarten ist (z. B. wenn der Verbraucher über die Benutzeroberfläche eine manuelle Suchfunktion ausführt), muss der Validator dem Benutzer deutlich signalisieren, dass eine unerwartete Unterbrechung aufgetreten ist, wenn die Reihenfolge nicht übereinstimmt. Dazu gehört auch die Überprüfung, ob die Positionswerte für einen bestimmten Merkle-Baum bei Null beginnen und für jeden folgenden Block um eins erhöht werden; entsprechend gibt der Positionswert immer an, welcher Block gerade wiedergegeben wird.

Bei Inhalten, die während der Wiedergabe über progressives Herunterladen validiert werden sollen, können die Blattknoten des Merkle-Baums an den Synchronisationspunkten der Videospur im „`mdat`“ ausgerichtet werden (z. B. den RAP-Punkten, den Random Access Points). Wenn die „`variableBlockSizes`“ so eingerichtet sind, dass eine solche Ausrichtung erreicht wird, kann die Validierung während der linearen Wiedergabe oder das Suchen nach der gewünschten Wiedergabezeit über dieselbe Sequenz erfolgen. Die gewünschten Blöcke werden abgerufen, validiert und die darin enthaltenen Spuren für die Wiedergabe ausgewählt.

Bei Inhalten, die absichtlich nicht wie vom Anspruchgenerator ursprünglich vorgesehen gerendert werden, z. B. beim Vor- und Zurückspulen oder bei der Wiedergabe mit einer anderen Geschwindigkeit, kann der Validator den Inhalt möglicherweise nicht validieren. In diesem Fall

Der Validator muss dem Benutzer deutlich signalisieren, dass der Inhalt während des entsprechenden Vorgangs nicht validiert werden kann.

Bei Inhalten mit C2PA ContentProvenanceBox, bei denen box_purpose auf „update presence" gesetzt ist, wird zunächst das aktive Manifest in der C2PA ContentProvenanceBox mit box_purpose auf „update" und dann in der C2PA ContentProvenanceBox mit box_purpose auf „original" gesucht. Befindet sich das aktive Manifest in der C2PA ContentProvenanceBox mit box_purpose auf „update", wird die übergeordnete Kette der Bestandteile zurückverfolgt (entweder in der C2PA ContentProvenanceBox mit box_purpose auf „update" oder „original", je nach Bedarf), bis das erste Nicht-Update-Manifest gefunden wird. Der BMFF-Hash dieses Manifestinhalts muss gemäß Abschnitt 9.2, „Hard Bindings", validiert werden. Das Hinzufügen einer C2PA ContentProvenanceBox mit box_purpose auf „update" sollte keinen Einfluss auf die Hash-Berechnung haben, da sie am Ende der Datei hinzugefügt wurde, ohne dass sich irgendwelche Offsets geändert haben.

Wenn die bmff-hash-map kein Feld „exclusions" enthält oder der Wert dieses Feldes nicht vom Typ Array mit mindestens einem Eintrag ist, wird das Manifest mit dem Fehlercode assertion.bmffHash.malformed abgelehnt.

Bestimmen Sie die Hash-Algorithmus-Kennung (oder den möglichen Fehlercode) gemäß dem in Abschnitt 15.4, „Bestimmen des Hash-Algorithmus", beschriebenen Verfahren.

Wenn der Endbyte-Offset eines Teilbereichs (Offset + Länge) größer ist als der Offset-Wert des nächsten Bereichs im Array oder wenn ein Offset- oder Längenwert negativ ist, wird das Manifest mit dem Fehlercode assertion.bmffHash.malformed abgelehnt. Der Fehlercode assertion.bmffHash.mismatch wird für alle anderen in diesem Abschnitt beschriebenen Fehler verwendet. Andernfalls fügt der Validator den Erfolgscode assertion.bmffHash.match zur Liste hinzu, die er schließlich zurückgibt.

Wenn der BMFF-Hash-Prozess einen Fehlercode assertion.bmffHash.mismatch erzeugt, muss der Validator nach dem Vorhandensein einer Multi-Asset-Hash-Assertion suchen. Ist eine vorhanden, darf der Fehlercode assertion.bmffHash.mismatch nicht ausgegeben werden, sondern die Multi-Asset-Hash-Assertion muss gemäß Abschnitt 15.12.4, „Validierung eines Multi-Asset-Hash", validiert werden; andernfalls muss das Manifest mit dem Fehlercode assertion.bmffHash.mismatch abgelehnt werden.

15.12.2.1. Nicht fragmentierte Asset unter Verwendung eines Merkle-Baums

Wenn das Merkle-Feld in der bmff-hash-map vorhanden ist, muss der Validator den Merkle-Baum validieren. Wenn fixedBlockSize und variableBlockSizes in bmff-merkle-map nicht vorhanden sind, wird die gesamte Nutzlast des mdat als einzelner Blattknoten für die Hash-Berechnung behandelt. Wenn fixedBlockSize vorhanden ist und variableBlockSizes nicht vorhanden ist, wird die Nutzlast des mdat in Blöcke fester Länge unterteilt, wobei jeder Block als Blattknoten behandelt wird. Wenn der letzte Block das Ende der mdat-Nutzlast überschreitet, sollte die Größe des letzten Blocks so festgelegt werden, dass sie sich nur bis zum Ende der mdat-Nutzlast erstreckt. Wenn variableBlockSize vorhanden ist und fixedBlockSizes nicht vorhanden ist, wird die Nutzlast des mdat in Größen unterteilt, die durch das Array von variableBlockSizes definiert sind. Wenn die Anzahl der Elemente nicht gleich count ist oder die Summe der Werte nicht gleich der Größe der Nutzlast des mdat ist, wird das Manifest mit dem Fehlercode assertion.bmffHash.malformed abgelehnt. Wenn fixedBlockSize und variableBlockSizes in bmff-merkle-map vorhanden sind, wird das Manifest mit dem Fehlercode assertion.bmffHash.malformed abgelehnt.

Wenn die Anzahl in der bmff-merkle-map gleich der Anzahl der Hash-Elemente in der bmff-merkle-map ist
und wenn der Hash des Blattknotens nicht mit dem Element der Hashes in der bmff-merkle-map übereinstimmt, wird das Manifest

mit dem Fehlercode assertion.bmffHash.mismatch abgelehnt. Wenn die Anzahl in der bmff-merkle-map kleiner ist als die Anzahl der Hash-Elemente in der bmff-merkle-map und wenn die zusätzliche UUID-C2PA-Box nicht wie in Abschnitt A.5.4 beschrieben vorhanden ist, „Zusatzfelder „c2pa" für große und fragmentierte Dateien", wird das Manifest mit dem Fehlercode assertion.bmffHash.malformed abgelehnt. Wenn der aus der zusätzlichen UUID-C2PA-Box und dem Blattknoten berechnete Hash nicht mit dem Element der Hashes in der bmff-merkle-map übereinstimmt, wird das Manifest mit dem Fehlercode assertion.bmffHash.mismatch abgelehnt. Wenn die Anzahl in der bmff-merkle-map größer ist als die Anzahl der Hash-Elemente in der bmff-merkle-map, wird das Manifest mit dem Fehlercode assertion.bmffHash.malformed abgelehnt.

15.12.2.2. Fragmentierte Assets unter Verwendung eines Merkle-Baums

Wenn das Merkle-Feld in der bmff-hash-map vorhanden ist, muss der Validator den Merkle-Baum validieren. Wenn die zusätzliche uuid C2PA-Box nicht wie in Abschnitt A.5.4, „Zusätzliche 'c2pa'-Boxen für große und fragmentierte Dateien", beschrieben vorhanden ist, muss das Manifest mit dem Fehlercode assertion.bmffHash.malformed abgelehnt werden. Wenn der aus der zusätzlichen UUID-C2PA-Box und dem Blattknoten berechnete Hash nicht mit dem Element der Hashes in der bmff-merkle-map übereinstimmt, ist nicht gleich, dann das Manifest soll sein abgelehnt mit einem Fehler Code von assertion.bmffHash.mismatch

15.12.3. Validierung eines allgemeinen Box-Hash

Sobald ein Standardmanifest (und seine Bindungen) gefunden wurde, muss die Liste der zu validierenden Boxen aus dem Feld „boxes" der Box-Map-Struktur extrahiert werden, die in der Assertion „c2pa.hash.boxes" gespeichert ist. Ist kein solches Feld vorhanden, muss das Manifest mit dem Fehlercode „assertion.boxesHash.malformed" abgelehnt werden.

Die Boxen müssen in der Ressource in derselben Reihenfolge erscheinen, in der sie im Boxen-Array erscheinen, einschließlich der Box, die das C2PA-Manifest enthält. Wenn andere Boxen in der Ressource vorhanden sind, wird das Manifest mit dem Fehlercode assertion.boxesHash.unknownBox abgelehnt. Wenn die Boxen in der falschen Reihenfolge erscheinen, wird das Manifest mit dem Fehlercode assertion.boxesHash.mismatch abgelehnt.

Wenn der Hash-Wert für eine Box nicht übereinstimmt und diese Box kein ausgeschlossenes Feld mit dem Wert „true" hat, wird das Manifest mit dem Fehlercode assertion.boxesHash.mismatch abgelehnt. Andernfalls fügt der Validator den Erfolgscode assertion.boxesHash.match zur Liste hinzu, die er schließlich zurückgibt.

Wenn der in einem alg-Feld angegebene Hash-Algorithmus nicht in der Liste der zulässigen oder veralteten Algorithmen in Abschnitt 13.1, „Hashing", aufgeführt ist oder ein alg-Feld weder in der Box-Map noch in einer bestimmten Box-Hash-Map aufgeführt ist, wird das Manifest mit dem Fehlercode algorithm.unsupported abgelehnt.

Wenn eine Box-Hash-Map im Array „boxes" kein Feld „names" enthält, wird das Manifest mit dem Fehlercode „assertion.boxesHash.malformed" abgelehnt.

Für jede Box, die im Array „names" und „boxes" aufgeführt ist, muss der angegebene Hash-Algorithmus über die Bytes der Box (zusammen mit allen zugehörigen Headern) berechnet werden. Wenn das Array „names" mehrere Einträge enthält, muss der Hash-Wert für diesen Bereich von Boxen vom Anfang der ersten Box (in diesem Bereich) bis zum Ende der letzten Box (in diesem Bereich) berechnet werden. Dies umfasst alle beliebigen Bytes, die zwischen den Boxen vorhanden sein können.

Wenn das Hash-Feld nicht vorhanden ist oder ein resultierender Hash nicht mit dem Wert des Hash-Feldes für diese Boxen übereinstimmt, wird das Manifest mit dem Fehlercode `assertion.boxesHash.mismatch` abgelehnt. Wenn der Box-Hash-Prozess einen Fehlercode `assertion.boxesHash.mismatch` erzeugt, sucht der Validator nach dem Vorhandensein einer Multi-Asset-Hash-Assertion. Ist eine vorhanden, muss sie wie in Abschnitt 15.12.4, „Validierung eines Multi-Asset-Hash“, beschrieben validiert werden. Ist keine vorhanden, muss das Manifest mit dem Fehlercode `assertion.boxesHash.mismatch` abgelehnt werden.

15.12.3.1. JPEG-Sonderbehandlung

Bei der Validierung eines JPEG muss ein Validierer überprüfen, ob jedes Feld, das mit der speziellen `C2PA`-Feldkennung gekennzeichnet ist, tatsächlich ein `APP11` ist, das einen Teil oder den gesamten C2PA-Manifest-Speicher enthält. Der C2PA-Manifest-Speicher wird dadurch identifiziert, dass es sich um eine JUMBF-Superbox mit der Bezeichnung `c2pa` und einer JUMBF-Typ-UUID von `63327061-0011-0010-8000-00AA00389B71` handelt, wie in Abschnitt 11.1.4.2, „Manifest-Speicher“, beschrieben.

Wenn ein `APP11` vorhanden ist, das nicht Teil des C2PA-Manifest-Speichers ist und nicht in der Liste der gehashten Boxen enthalten ist, wird das Manifest mit dem Fehlercode `assertion.boxesHash.unknownBox` abgelehnt.

15.12.3.2. Spezielle Behandlung von Schriftarten

Bei der Validierung einer Schriftart muss ein Validierer überprüfen, ob das Feld, das der `C2PA`-Tabelle der Schriftart entspricht, vorhanden ist, und feststellen, ob es ein eingebettetes Manifest, eine Remote-Manifest-URI oder beides enthält.

Wenn Schriftarttabellen vorhanden sind, die nicht von einem Feld abgedeckt sind, muss das Manifest mit dem Fehlercode `assertion.boxesHash.unknownBox` abgelehnt werden.

15.12.4. Validierung eines Multi-Asset-Hash

Wenn die Standardvalidierung der Hard-Binding-Assets fehlschlägt und das Asset eine Multi-Asset-Hash-Assertion enthält, muss der Validator mit der Validierung der Multi-Asset-Hash-Assertion fortfahren. Wenn mehr als eine Multi-Asset-Hash-Assertion vorhanden ist, muss das Manifest mit dem Fehlercode `assertion.multiAssetHash.malformed` abgelehnt werden.

Die Validierung der Multi-Asset-Hash-Assertion (`c2pa.hash.multi-asset`) erfolgt durch Iteration über das Array der `Teile` in der `Multi-Asset-Hash-Map`. Wenn das Feld „`parts`“ nicht vorhanden ist oder einen Wert enthält, der ein leeres Array ist, wird das Manifest mit dem Fehlercode `assertion.multiAssetHash.malformed` abgelehnt.

Für jeden Teil muss der Validator sicherstellen, dass er sowohl einen gültigen `Locator` als auch ein gültiges hashAssertion-Feld enthält. Fehlt eines dieser Felder, wird das Manifest mit dem Fehlercode `assertion.multiAssetHash.malformed` abgelehnt.

Wenn es sich bei dem Locator um `einen Byte-Offset-Locator` handelt, muss der Validator sicherstellen, dass die Felder „`byteOffset`“ und „`length`“ vorhanden, nicht negativ und nicht größer als die Gesamtlänge des Assets sind. Fehlt eines dieser Felder, ist es negativ oder zu groß, wird das Manifest mit dem Fehlercode „`assertion.multiAssetHash.`malformed“ abgelehnt.

Wenn der Locator durch einen `bmffBox` dargestellt wird, muss der Validator sicherstellen, dass die angegebene Box im Asset vorhanden ist

Asset vorhanden ist. Wenn die Box vorhanden ist nicht vorhanden ist, dann das Manifest soll abgelehnt werden mit einem Fehlercode
`assertion.multiAssetHash.malformed`.

Bei Vorliegen eines gültigen Lokators und Hashs versucht der Validator, das Teil anhand der Lokatorinformationen zu finden. Ist es nicht vorhanden und ist das `optionale` Feld entweder nicht vorhanden oder mit dem Wert „`false`“ versehen, wird das Manifest mit dem Fehlercode `assertion.multiAssetHash.missingPart` abgelehnt. Ist das `optionale` Feld mit dem Wert „`true`“ vorhanden, überspringt der Validator dieses Teil und fährt mit dem nächsten Teil fort.

HINWEIS	Das Verwerfen bestimmter Teile kann dazu führen, dass ein Validator nicht mehr in der Lage ist, das Manifest eindeutig zu identifizieren. verbleibenden Teile eindeutig identifizieren. In den meisten Fällen können nur ein oder mehrere Teile am Ende der Datei, nicht jedoch Teile in der Mitte, effektiv verworfen werden.

Wenn sich die gefundenen Teile überschneiden oder insgesamt nicht jedes Byte des Assets abdecken, wird das Manifest mit dem Fehlercode `assertion.multiAssetHash.malformed` abgelehnt.

Für jeden gefundenen Teil berechnet der Validator den Hash des Teils unter Verwendung des angegebenen Algorithmus und der angegebenen Methodik (d. h. Daten-Hash, General-Box-Hash oder BMFF-Hash) über die Bytes des Teils. Wenn der resultierende Hash nicht mit dem Wert übereinstimmt, der in der vom Feld „`hashAssertion`” referenzierten Hard-Binding-Assertion vorhanden ist, wird das Manifest mit dem Fehlercode „`assertion.multiAssetHash.mismatch`” abgelehnt.

Wenn die Hash-Assertion für jedes gefundene Teil erfolgreich validiert wurde, muss der Validator den Erfolgscode `assertion.multiAssetHash.match` aufzeichnen und darf keine Fehlercodes aufzeichnen, die mit der Hard Binding-Assertion des Assets verbunden sind.

15.12.5. Validieren eines Sammlungsdaten-Hash

15.12.5.1. Allgemeines

Die Validierung einer Hash-Assertion für Sammlungsdaten (`c2pa.hash.collection.data`), die in einem Standardmanifest gefunden wurde, erfolgt durch Iteration über das Array von `URIs` in der `collection-data-hash-map`. Ist kein URI-Feld vorhanden, wird das Manifest mit dem Fehlercode `assertion.collectionHash.malformed` abgelehnt.

Der zu verwendende spezifische Hash-Algorithmus wird anhand des Werts des Felds „`alg`“ bestimmt und gemäß Abschnitt 15.4.3, „Algorithmusvalidierung“, verarbeitet. Ist kein Feld „`alg`“ vorhanden, wird das Manifest mit dem Fehlercode „`assertion.collectionHash.`malformed“ abgelehnt.

Für jede `uri-hashed-data-map` im Array `uris` muss der Validator sicherstellen, dass sie sowohl ein `uri-` als auch ein hash-Feld enthält. Fehlt eines dieser Felder, wird das Manifest mit dem Fehlercode `assertion.collectionHash.malformed` abgelehnt.

Um potenzielle Sicherheitsrisiken zu vermeiden, muss ein Validator die URIs (d. h. den Wert des Feldes „`uri`“) vor der Verwendung validieren und sicherstellen, dass weder `.` noch `..` als Teil der URI vorkommen. Wenn eines dieser Zeichen in einer URI gefunden wird, muss das Manifest mit dem Fehlercode „`assertion.collectionHash.invalidURI`“ abgelehnt werden.

Für das aus der URI abgerufene Asset muss dessen Hash unter Verwendung des angegebenen Algorithmus über alle Bytes seiner Daten berechnet werden. Wenn der resultierende Hash nicht mit dem Wert des Hash-Feldes übereinstimmt, muss das Manifest mit dem Fehlercode `assertion.collectionHash.mismatch` abgelehnt werden. Andernfalls muss der Validator den Erfolgscode `assertion.collectionHash.match` zu der Liste hinzufügen, die er schließlich zurückgibt.

Wenn in der Hash-Assertion der Sammlungsdaten Dateien aufgeführt sind, die vom Validator nicht gefunden werden, wird das Manifest mit dem Fehlercode `assertion.collectionHash.incorrectFileCount` abgelehnt.

15.12.5.2. Extras für ZIP

In einer ZIP-Datei mit einem zugehörigen C2PA-Manifest enthält der Hash der Sammlungsdaten das zusätzliche Feld `zip_central_directory_hash`. Wie bereits beschrieben, enthält dieses Feld einen Hash jedes „zentralen Verzeichnis-Headers" im zentralen Verzeichnis der ZIP-Datei sowie den „Ende des zentralen Verzeichnis-Datensatzes" (der letzte Teil einer ZIP-Datei). Der für dieses Feld verwendete Hash-Algorithmus ist derselbe wie der für das Hash-Feld in der `c2pa.hash.collection.`data-Assertion verwendete.

Bei der Validierung einer ZIP-Datei muss der Validator überprüfen, ob das Feld `zip_central_directory_hash` vorhanden ist und ob der Hash des ZIP-Zentralverzeichnisses und des „Ende des Zentralverzeichnisdatensatzes" mit seinem Wert übereinstimmt. Wenn der Hash nicht übereinstimmt, wird das Manifest mit dem Fehlercode `assertion.collectionHash.mismatch` abgelehnt.

Kapitel 16. Benutzererfahrung

16.1. Ansatz

Die C2PA beabsichtigt, klare Empfehlungen und Leitlinien für Implementierer von Herkunfts-fähigen Benutzererfahrungen (UX) bereitzustellen. Die Entwicklung dieser Empfehlungen ist ein fortlaufender Prozess, an dem verschiedene Interessengruppen beteiligt sind. Die Ergebnisse sorgen für ein Gleichgewicht zwischen Einheitlichkeit und Vertrautheit einerseits und Nutzen und Flexibilität für Benutzer über Kontexte, Plattformen und Geräte hinweg andererseits. Diese Empfehlungen finden Sie im Dokument „Leitlinien zur Benutzererfahrung“.

16.2. Grundsätze

Die UX-Empfehlungen zielen darauf ab, Best Practices für die Darstellung der C2PA-Provenienz gegenüber Verbrauchern zu definieren. Die Empfehlungen sollen standardisierte, leicht erkennbare Erfahrungen beschreiben, die

- den Erstellern von Assets die Möglichkeit bieten, Informationen und den Verlauf der von ihnen erstellten Inhalte zu erfassen, und
- Bereitstellung von Informationen und Hintergrundwissen für Nutzer von Assets zu den Inhalten, mit denen sie sich beschäftigen, damit sie deren Herkunft nachvollziehen und entscheiden können, inwieweit sie diesen vertrauen können.

Benutzeroberflächen, die für die Nutzung der C2PA-Herkunft konzipiert sind, müssen sich am Kontext der Ressource orientieren. Wir haben vier primäre Benutzergruppen und eine Reihe von Kontexten untersucht, in denen C2PA-Assets vorkommen. Diese Benutzergruppen wurden in den C2PA-Leitprinzipien als Verbraucher, Ersteller, Herausgeber und Prüfer (oder Ermittler) definiert. Um den Anforderungen jeder dieser Gruppen in gängigen Kontexten gerecht zu werden, werden für viele gängige Fälle beispielhafte Benutzeroberflächen vorgestellt. Dabei handelt es sich um Empfehlungen, nicht um Vorschriften, und wir gehen davon aus, dass sich bewährte Verfahren weiterentwickeln werden.

16.3. Offenlegungsstufen

Da der vollständige Satz von C2PA-Daten für ein bestimmtes Asset für einen Nutzer überwältigend sein kann, beschreiben wir vier Stufen der schrittweisen Offenlegung, die als Leitfaden für die Gestaltung dienen:

- Stufe 1: Ein Hinweis darauf, dass C2PA-Daten vorhanden sind, und deren kryptografischer Validierungsstatus.
- Stufe 2: Eine Zusammenfassung der für eine bestimmte Ressource verfügbaren C2PA-Daten. Diese Stufe sollte genügend Informationen für den jeweiligen Inhalt, Benutzer und Kontext bereitstellen, damit der Verbraucher ausreichend nachvollziehen kann, wie die Ressource zu ihrem aktuellen Zustand gelangt ist.
- Stufe 3: Eine detaillierte Anzeige aller relevanten Herkunftsdaten. Beachten Sie, dass die Relevanz bestimmter Elemente gegenüber anderen kontextabhängig ist und vom UX-Implementierer festgelegt wird.
- Stufe 4: Für anspruchsvolle forensische Untersuchungen wird ein Tool empfohlen, das alle granularen Details von Signaturen und Vertrauenssignalen aufzeigen kann.

16.4. Öffentliche Überprüfung, Feedback und Weiterentwicklung

Das Team, das die UX-Empfehlungen verfasst, ist sich seiner Grenzen und potenziellen Verzerrungen bewusst und erkennt an, dass Feedback, Überprüfung, Benutzertests und kontinuierliche Weiterentwicklung eine wichtige Voraussetzung für den Erfolg sind. Die Empfehlungen werden daher ein sich weiterentwickelndes Dokument sein, das auf realen Erfahrungen mit dem Einsatz von C2PA UX in einer Vielzahl von Anwendungen und Szenarien basiert.

Kapitel 17. Informationssicherheit

17.1. Bedrohungen und Sicherheitsaspekte

Dieser Abschnitt enthält eine Zusammenfassung der Überlegungen und Prozesse zur Informationssicherheit für die in der C2PA-Kernspezifikation beschriebene Technologie. Ausführlichere Informationen werden in zukünftigen Veröffentlichungen von C2PA-Materialien, einschließlich des Leitfadens, bereitgestellt.

17.1.1. Kontext

Die Informationssicherheit ist ein zentrales Anliegen von C2PA. C2PA unterhält ein Bedrohungsmodell und Sicherheitsüberlegungen für die C2PA-Spezifikation. Diese Bemühungen ergänzen andere sicherheitsbezogene Arbeiten innerhalb von C2PA. Die zugehörige Dokumentation befindet sich derzeit in Entwicklung und ist unter Sicherheitsüberlegungen zu finden.

Die C2PA entwickelt derzeit eine Dokumentation zu Sicherheitsaspekten, die Folgendes umfasst:

- Eine Zusammenfassung der relevanten Sicherheitsmerkmale der C2PA-Technologie
- Sicherheitsüberlegungen für den praktischen Einsatz der C2PA-Technologie
- Bedrohungen für die C2PA-Technologie und deren jeweilige Behandlung, einschließlich Gegenmaßnahmen

17.1.2. Übersicht über den Prozess der Bedrohungsmodellierung

Die C2PA integriert Sicherheit bereits während der Entwicklung in unsere Designs, geht jedoch davon aus, dass das Sicherheitsdesign und die Bedrohungsmodellierung mit der Weiterentwicklung des Systems, des Ökosystems und der Bedrohungslandschaft fortgesetzt werden.

Zu diesem Zweck nutzt die C2PA einen fokussierten Prozess zur Bedrohungsmodellierung, um die Entwicklung eines starken Sicherheits- und Datenschutzdesigns zu unterstützen. Die Ergebnisse dieser Bemühungen unterstützen direkt die Entwicklung einer Dokumentation expliziter Bedrohungen und Sicherheitsüberlegungen, fördern aber auch das Sicherheitsdenken während des gesamten Designprozesses.

Der Prozess der Bedrohungsmodellierung kombiniert synchrone (Live-)Sitzungen zur Bedrohungsmodellierung, an denen Fokusgruppen aus Fachexperten (SMEs) teilnehmen, mit der asynchronen Entwicklung von Inhalten. Die Teilnehmerzahl jeder synchronen Sitzung wird gering gehalten, um effiziente Diskussionen zu fördern, aber alle Mitglieder der C2PA haben die Möglichkeit, über beide Modalitäten teilzunehmen.

Wie andere Sicherheitsaktivitäten auch, erwarten wir, dass sich unser Prozess der Bedrohungsmodellierung mit dem C2PA-Ökosystem weiterentwickelt. Die Prozessdokumentation ist eher als Leitfaden zu verstehen und nicht als strenge Vorgabe für die Funktionsweise der Bedrohungsmodellierung innerhalb der C2PA.

17.1.2.1. Referenzen

Eine Vielzahl von Referenzen und Erfahrungen fließen in die Bedrohungsmodellierung und damit verbundene Sicherheitsmaßnahmen für die C2PA ein. Dieser Abschnitt enthält eine Auswahl öffentlich zugänglicher Dokumente als Referenz.

- IETF zu Sicherheitsaspekten

- IETF zu Datenschutzaspekten (Richtlinien)
- W3C-Fragebogen zur Selbstbewertung von Sicherheit und Datenschutz
- OAuth2-Bedrohungsmodell (Beispiel)
- Bedrohungsmodellierung: Design für Sicherheit
- OWASP-Bedrohungsmodellierung
- Microsoft-Bedrohungsmodellierung

17.2. Schäden, Missbrauch und Zweckentfremdung

17.2.1. Einleitung

Die C2PA-Leitprinzipien legen fest, dass C2PA-Spezifikationen kritisch auf potenziellen Missbrauch oder Fehlgebrauch des Frameworks hin überprüft werden müssen, um unbeabsichtigte Schäden, Menschenrechtsverletzungen oder unverhältnismäßige Risiken für schutzbedürftige Gruppen weltweit zu vermeiden.

Um sicherzustellen, dass die C2PA diesen Aspekt ihrer Grundsätze erfüllt, zielt die Bewertung von Schäden, Missbrauch und Missbrauch darauf ab, potenzielle Probleme während der Entwicklung der Spezifikationen und bei der anschließenden Umsetzung zu identifizieren und zu beheben.

Darüber hinaus werden die Spezifikationen überprüft, um:

- potenzielle Missbräuche und Fehlverwendungen zu antizipieren und zu mindern;
- häufige Datenschutzbedenken der Nutzer zu berücksichtigen; und
- die Bedürfnisse von Nutzern und Interessengruppen weltweit zu berücksichtigen.

17.2.2. Überlegungen

Die Bewertung von Schäden, Missbrauch und Fehlgebrauch ist ein fortlaufender Prozess. Die in der Dokumentation zur Schadensmodellierung dargestellten Informationen sollten nicht als Endergebnis einer umfassenden Bewertung betrachtet werden, sondern als Grundlage für fortlaufende Diskussionen, die sich auf die betroffenen Gemeinschaften konzentrieren und darauf abzielen, potenziellen Missbrauch und Fehlgebrauch zu mindern und die Menschenrechte zu schützen.

Der Ansatz umfasst zwei entscheidende Aspekte:

Laufend

> Die Bewertung von Schäden, Missbrauch und Missbrauch begleitet notwendigerweise die Konzeption und Entwicklung sowie die Implementierungs- und Nutzungsphasen der C2PA, indem sie kontinuierlich in den Spezifikationsentwicklungsprozess, die Implementierungs- und Benutzererfahrungsleitfäden, Sensibilisierungsmaßnahmen, die Governance der Koalition und möglicherweise multilaterale Kooperationen zur Förderung eines vielfältigen C2PA-Ökosystems, das einem breiten Spektrum globaler Kontexte dient, einfließt.

Multidisziplinär und vielfältig

Die Bewertung von Schäden, Missbrauch und Missbrauch sollte eine gemeinsame Anstrengung sein, an der multidisziplinäre Experten und ein breites Spektrum von Interessengruppen mit praktischer und technischer Erfahrung in diesen Fragen aus verschiedenen geografischen Regionen, mit unterschiedlichem kulturellem Hintergrund und individueller Identität beteiligt sind.

17.2.3. Bewertung

Die Schadensmodellierung konzentriert sich auf die Analyse, wie sich ein soziotechnisches System negativ auf Nutzer, andere Interessengruppen oder die Gesellschaft insgesamt auswirken oder anderweitig Strukturen der Ungerechtigkeit, Bedrohungen der Menschenrechte oder unverhältnismäßige Risiken für schutzbedürftige Gruppen weltweit schaffen oder verstärken könnte. Der Prozess der Schadensmodellierung erfordert systematisch die Kombination von Wissen über eine Systemarchitektur und ihre Nutzerfreundlichkeit mit historischen und kontextbezogenen Erkenntnissen über die Auswirkungen ähnlicher bestehender Systeme auf verschiedene soziale Gruppen sowie die partizipative Konsultation einer Reihe von Gemeinschaften, die von dem System betroffen sein könnten. Diese kombinierten Informationen bilden den Rahmen für die Fähigkeit, Schäden zu antizipieren und proaktiv Reaktionen zu identifizieren.

Die Dokumentation zur Schadensmodellierung beschreibt den Rahmen und den bisher durchgeführten Prozess, gefolgt von der Methodik, einem Überblick über die Bewertung, einem Entwurf für die öffentliche Überprüfung und Rückmeldung sowie den Sorgfaltspflichten, die zur Begleitung der Version 1.0 dieser Spezifikationen, ihrer Umsetzung und Weiterentwicklung entwickelt werden.

17.2.4. Sorgfaltspflichten

Die Bewertung von Schäden, Missbrauch und Zweckentfremdung hat die Entwicklung der technischen Spezifikationen der C2PA sowie der dazugehörigen Dokumentation beeinflusst und sollte dies auch weiterhin tun:

- Leitfaden für Implementierer
- Leitfaden zur Benutzererfahrung
- Sicherheitsaspekte
- Erläuterungen

Darüber hinaus sollte die Bewertung von Schäden, Missbrauch und Missbrauch in die Leitung der Koalition einfließen und als Leitfaden für eine mögliche multilaterale Zusammenarbeit zur Förderung eines vielfältigen C2PA-Ökosystems dienen, das die Optimierung der Vorteile in Bezug auf Vertrauen in die Medien, Benutzerkontrolle und Transparenz vorantreibt, die zur Entwicklung der C2PA-Spezifikationen geführt haben.

Kapitel 18. C2PA-Standardaussagen

18.1. Einleitung

Dieser Abschnitt des Dokuments listet die Standardaussagen für die Verwendung durch C2PA-Implementierungen auf und beschreibt deren Syntax, Verwendung usw. Der Einfachheit halber wurden alle Beispiel-JUMBF-URIs zu Illustrationszwecken gekürzt – in den tatsächlichen Daten sind vollständige URIs erforderlich.

Alle Assertions müssen eine Kennzeichnung gemäß Abschnitt 6.2, „Kennzeichnungen", aufweisen und gemäß Kapitel 5, „Versionierung", versioniert sein.

*Version*ierung

Alle C2PA-standardisierten Assertions verwenden den JSON-JUMBF-Inhaltstyp, den CBOR-JUMBF-Inhaltstyp oder den Embedded-File-Inhaltstyp aus ISO 19566-5:2023. Entitätsspezifische Assertions können einer dieser Typen sein, einer der anderen JUMBF-Inhaltstypen aus ISO 19566-5:2023, Anhang B (z. B. XML) oder sie können einen eigenen Typ erstellen (gemäß den Anweisungen in ISO 19566-5:2023, Tabelle B.1). Der Codestream-Inhaltstyp darf nicht für eine C2PA-Assertion verwendet werden.

Sofern nicht anders angegeben, müssen alle in diesem Standardsatz von Assertions dokumentierten Assertions als CBOR serialisiert werden. Alle als CBOR serialisierten Assertions müssen den Core Deterministic Encoding Requirements von CBOR (siehe RFC 8949, Abschnitt 4.2.1) entsprechen, und ihre Schemata müssen unter Verwendung einer CDDL-Definition definiert werden.

HINWEIS	Alle CDDLs gelten als nicht normativ.

Für diejenigen, die mit JSON definiert sind, müssen ihre Schemata mit der neuesten Version von JSON Schema definiert werden.

18.2. Regionen von Interesse

18.2.1. Beschreibung

In einigen Anwendungsfällen kann eine bestimmte Aussage, wie z. B. eine Aktionsaussage, nur für einen bestimmten Teil eines Assets und nicht für das gesamte Asset relevant sein. In diesen Fällen muss es eine Möglichkeit geben, diesen Bereich zu beschreiben – sei es zeitlich, räumlich, textuell oder eine Kombination davon. Eine Bereichsdefinition dient diesem Zweck.

18.2.2. Allgemein

Der wichtigste Teil der Regionsdefinition ist das Feld „`range`", das verwendet wird, um einen zeitlichen Bereich, einen räumlichen Bereich, einen Frame-Bereich, einen Textbereich oder eine Kombination davon für die Region zu beschreiben.

HINWEIS	Die Spezifikation erlaubt zwar die Angabe einer Kombination von Bereichen, es ist jedoch nicht definiert, wie ein Manifest Verbraucher werden sie nutzen. Es wird erwartet, dass sich die User Experience Task Force der C2PA in Zukunft damit befassen wird.

Eine `Region` kann auch eines oder mehrere der folgenden gemeinsamen Felder enthalten:

name

Eine Freitextzeichenfolge, die einen für Menschen lesbaren Namen für die Region darstellt, der in einer Benutzeroberfläche verwendet werden kann.

Bezeichner

Eine Freitextzeichenfolge, die einen maschinenlesbaren, für diese Aussage eindeutigen Identifikator für die Region darstellt.

Typ

Ein Wert aus einem kontrollierten Vokabular wie https://cv.iptc.org/newscodes/imageregiontype/ oder ein entitätsspezifischer Wert (z. B. `com.litware.newType`), der den Typ der durch eine Region dargestellten Dinge repräsentiert.

Beschreibung

Eine Freitextzeichenfolge.

Ältere Versionen dieser Spezifikation enthielten ein Rollenfeld. Dieses Feld ist veraltet und sollte bei der Generierung einer Region von Interesse nicht mehr verwendet werden.

18.2.2.1. Bereiche

Alle Bereiche bestehen aus einem Typfeld, dessen Wert entweder „spatial“, „temporal“, „frame“, „textual“ oder „identified“ lautet. Darüber hinaus muss es eines der folgenden Felder enthalten, dessen Daten ein Objekt sind, das aus den spezifischen Daten für diesen Bereich besteht:

- `shape` (für räumlich);
- `time` (für temporal);
- `frame` (für temporal oder textual);
- `text` (für textuell);
- `item` (für spezifisch identifizierte Elemente).

18.2.2.2. Räumlich

Räumliche Bereiche werden mithilfe eines Formobjekts beschrieben. Eine `Form` kann verwendet werden, um ein Rechteck, einen Kreis oder ein Polygon darzustellen. Sie ist der Region Boundary Structure (Gebietsgrenzstruktur) der IPTC nachempfunden.

18.2.2.3. Zeit

Zeitbereiche werden mithilfe eines Zeitobjekts beschrieben, das einen Bereich von einer Startzeit bis zu einer Endzeit darstellt. Zeiten werden entweder mithilfe der normalen Wiedergabezeit (`npt`) gemäß RFC 2326 (wie in der W3C-Spezifikation für Medienfragmente empfohlen) oder einer „Wanduhrzeit” gemäß dem Internetprofil von ISO 8601 gemäß RFC 3339 beschrieben.

HINWEIS Die „Wall Clock Time” ist nützlich in Szenarien, in denen das Medienelement eine Aktivität darstellt, die zu einem bestimmten Datum und Zeitpunkt stattgefunden hat, wie z. B. eine Nachrichtensendung oder eine Live-Veranstaltung.

Wenn kein Typfeld angegeben ist, wird davon ausgegangen, dass der Bereich im npt-Format vorliegt. Wenn kein Startfeld angegeben ist, beginnt der Bereich am Anfang des Assets. Wenn kein Endfeld angegeben ist, endet der Bereich am Ende des Assets. Wenn keines von beiden

angegeben, umfasst der Bereich das gesamte Asset.

18.2.2.4. Frames

Ein Frame-Objekt definiert einen Bereich anhand des Start- und End-Frames oder der Start- und End-Seite (inklusive). Wenn kein `Start` angegeben ist, beginnt der Bereich am Anfang des Assets. Wenn kein `Ende` angegeben ist, endet der Bereich am Ende des Assets. Wenn weder Start noch Ende angegeben sind, umfasst der Bereich das gesamte Asset.

Frames werden als einzelne Ordnungszahlen dargestellt, wobei `0` der erste Frame ist.

Während Frames in der Regel zur Darstellung von Seitenzahlen eines Dokuments, beispielsweise eines PDF-Dokuments, verwendet werden, können sie auch in anderen Medientypen wie Animationen, Videos und Audiodateien zum Einsatz kommen. Es wird empfohlen, bei Medientypen, die sich mit bestimmten Zeitabschnitten befassen, nach Möglichkeit stattdessen Zeitbereiche zu verwenden.

18.2.2.5. Text

Ein Textobjekt definiert einen Bereich mithilfe eines oder mehrerer URL-Fragmentidentifikatoren, wie vom W3C Web Annotation Fragment Selector definiert. Es kann den Bereich auch mithilfe von Offsets zu den Start- und Endzeichen (inklusive) verfeinern. Wenn kein `Start` angegeben ist, beginnt der Bereich am Anfang des Fragments. Wenn kein `Ende` angegeben ist, endet der Bereich am Ende des Fragments. Wenn beides nicht angegeben ist, umfasst der Bereich das gesamte Fragment.

Bei einzelner Verwendung repräsentiert der Fragmenteintrag der `Text-Selektor-Zuordnung` den gesamten angegebenen Textbereich. Die `Text-Selektor-Bereichs-Zuordnung` unterstützt jedoch ein Paar von Text-Selektor-Zuordnungsobjekten. Der Wert von `selector` ist der Anfang des Bereichs (oder dessen Gesamtheit, wenn kein end-Eintrag vorhanden ist), und der Wert von `end` (falls vorhanden) stellt das Ende eines zusammenhängenden Bereichs dar. Darüber hinaus können mehrere Paare verwendet werden, um einen nicht zusammenhängenden Bereich darzustellen.

18.2.2.6. Identifiziert

Ein Item-Objekt definiert eine Medienspur, ein Medienelement oder ein anderes diskretes Inhaltselement im Asset, sodass der Claim-Generator Aussagen machen kann, die nur für einen Teil des Inhalts gelten, der im Dateicontainer des Assets enthalten ist. Es könnte beispielsweise verwendet werden, um anzugeben, dass nur die Audiospur einer Videodatei relevant ist.

Das Medien- oder Inhaltselement wird durch eine Kennzeichnungszeichenfolge identifiziert, deren Wert dem typischen Benennungsschema für die Elementidentifizierung in diesem bestimmten Containerformat entsprechen sollte. Beispielsweise sollte der Wert `der Kennzeichnung` für MP4-Dateien `„track_id“` und für HEIF-Dateien `„item_ID“` lauten. Der Wert der `Kennzeichnung` wird dann im Feld `„value“` angegeben. Beispielsweise würde ein `Wert` von 2 mit einer `Kennung` von `track_id` in einem MP4-Videodateicontainer eine Aussage in Bezug auf die zweite Medienspur in der Datei (die die Audiospur sein könnte) anzeigen.

Eine weitere Verwendung für identifizierte Bereiche ist die Angabe einer bestimmten Region durch einen bekannten semantischen Wert. Beispielsweise könnte das Foundational Model of Anatomy verwendet werden, um eine bestimmte Region des menschlichen Körpers zu identifizieren. In einem solchen Fall muss die `Kennung` die URL oder URI sein, unter der das Schema zu finden ist (allerdings nicht unbedingt direkt zu einem maschinenlesbaren Schema).

18.2.3. Schema

Das Schema für diesen Typ wird durch die Region-Map-Regel in der folgenden CDDL-Definition definiert:

```
region-map = {
     „region”:           [1* $range-map],             ; Definition des Bereichs, ein oder mehrere Bereiche
     ? „name”: tstr .size (1..max-tstr-length), ; eine Freitextzeichenfolge, die einen für Menschen lesbaren Namen
für die Region darstellt, der in einer Benutzeroberfläche verwendet werden könnte.
     ? „identifier”: tstr .size (1..max-tstr-length), ; eine Freitextzeichenfolge, die einen
maschinenlesbaren, für diese Aussage eindeutigen Identifikator für die Region darstellt.
     ? „type”:      tstr .size (1..max-tstr-length),           ; aus einer kontrollierten Liste
     ? „role”:      $role-choice,       ; VERALTET
     ? „description“: tstr .size (1..max-tstr-length), ; menschenlesbare Beschreibung der Region
     ? „metadata”: $assertion-metadata-map, ; zusätzliche Informationen zur Assertion
}

$range-choice /= „spatial“             ; ein durch einen physischen Bereich identifizierter Bereich
$range-choice /= „temporal”            ; ein durch einen Zeitraum identifizierter Bereich
$range-choice /= „frame“               ; ein Bereich, der durch eine Reihe von Frames oder Seiten identifiziert wird
$range-choice /= „textual”             ; ein Bereich, der durch einen Textbereich identifiziert wird
$range-choice /= "identified"          ; ein Bereich, der durch eine bestimmte Kennung und einen bestimmten Wert
identifiziert wird

range-map = {
     „type”:         $range-choice,          ; entweder „spatial”, „temporal”, „frame”, „textual” oder
„identified”
      ? „shape”:    $shape-map,              ; Beschreibung der Form eines räumlichen Bereichs
      ? „Zeit“:     $time-map,               ; Beschreibung der zeitlichen Grenzen eines zeitlichen Bereichs
      ? „frame“:    $frame-map,              ; Beschreibung der Rahmengrenzen eines zeitlichen
Bereich
     ? „text”:      $text-map,              ; Beschreibung der Grenzen eines Textbereichs
     ? „item”:      $item-map,              ; Beschreibung der Grenzen eines identifizierten Bereichs
}

Koordinatenkarte = {
     „x”: float,         ; Koordinate entlang der x-Achse
     "y": float,         ; Koordinate entlang der y-Achse
}

$shape-choice /= "rectangle"          ; eine rechteckige Form
$shape-choice /= "circle"             ; eine kreisförmige Form
$shape-choice /= "polygon"            ; eine polygonale Form

$unit-choice /= "pixel"          ; Einheiten sind in Pixeln angegeben
$unit-choice /= "percent"        ; Einheiten sind in Prozent der Gesamtgröße


shape-map = {
     „Typ“:         $shape-choice,               ; entweder „Rechteck“, „Kreis“ oder „Polygon“
     „Einheit“:     $unit-choice,                ; entweder „pixel“ oder „prozent“
     „origin“:      $Koordinatenkarte,           ; Start-/Ursprungskoordinate der Form.
     ? „width”: Float,                           ; Breite für Rechtecke, Durchmesser für Kreise (wird
bei Polygonen ignoriert)
     ? „height“: float                           ; Höhe für Rechtecke
     ? „inside“: bool,                           ; innerhalb oder außerhalb der Form, Standardwert ist `true`
     ? „vertices”:       [1* $Koordinatenkarte]       ; verbleibende Punkte/Eckpunkte des Polygons
}

; npt- und utc-Start- und Endzeiten haben unterschiedliche Regex-Formate
time-map = npt-time-map / wall-clock-time-map
```

```
npt-time-map = {
     ? „type“:      „npt”; falls nicht vorhanden, „npt”-Zeitkarte annehmen
     ? „start”: tstr .regexp „^(?:(?:([01]?\d|2[0-3]):)?([0-5]?\d):)?([0-5]?\d)(\.(\d{1,9}))?$”,
                                ; Startzeit (oder Beginn des Assets, falls nicht vorhanden).
     ? „end”:                   ; Endzeit (oder Ende des Assets, falls nicht
vorhanden).
}

wall-clock-time-map = {
     „type”:        "wallClock"; muss für die Zeitzuordnung "wall-clock" vorhanden sein
     ? „start”: tstr .regexp „^(\d{4})-(\d{2})-(\d{2})T(\d{2}):(\d{2}):(\d{2})(\.\d+)?(([+-
]\d{2}:\d{2})|Z)$", ; Startzeit (oder Beginn des Assets, falls nicht vorhanden).
     ? „end”:
]\d{2}:\d{2})|Z)", ; Endzeit (oder Ende des Assets/Live-Edge, falls nicht vorhanden).
}

; dies kann entweder für Frames eines Videos oder für Seiten eines Dokuments
verwendet werden frame-map = {
     ? „start”: int,          ; Startframe (oder Anfang des Assets, falls nicht vorhanden).
     ? „end”:       int       ; Endframe (oder Ende des Assets, falls nicht vorhanden).
}

; dies ist dem W3C-Webannotations-Selektormodell nachempfunden text-
selector-map = {
     „Fragment“:         tstr,     ; Fragmentbezeichner gemäß RFC3023 oder ISO 32000-2, Anhang O
     ? „start”:          int,      ; Startzeichen-Offset (oder Anfang des Fragments, falls nicht
vorhanden).
     ? „end”:            int       ; Endzeichen-Offset (oder Ende des Fragments, falls nicht vorhanden).
}

; eine oder zwei Text-Selektor-Zuordnungen zur Identifizierung des
Bereichs text-selector-range-map = {
     „selector”:         $text-selector-map,           ; Start- (oder einziger) Textauswahl
     ? „end”:            $text-selector-map            ; falls vorhanden, stellt das Ende des Textbereichs
dar
}

text-map = {
     „selectors“: [1* $text-selector-range-map]; Array von (möglicherweise diskontinuierlichen) Textbereichen
}

item-map = {
     „identifier”:       tstr .size (1..max-tstr-length),           ; der containerspezifische Begriff, der
zur Identifizierung von Elementen verwendet wird, z. B. „track_id” für MP4 oder „item_ID” für HEIF
     „value”:            tstr .size (1..max-tstr-length),           ; der Wert des Identifikators, z. B. würde
ein Wert von „2” für einen Identifikator von „track_id” den Titel 2 des Assets bedeuten
}

; Diese Werte sind veraltet
$role-choice /= „c2pa.areaOfInterest“ ; beliebiger Bereich, der identifiziert werden sollte
$role-choice /= „c2pa.cropped”                 ; dieser Bereich ist alles, was nach einer Zuschneideaktion übrig
bleibt
$role-choice /= "c2pa.edited"                  ; dieser Bereich wurde bearbeitet
$role-choice /= "c2pa.placed"                  ; der Bereich, in dem eine Zutat platziert/hinzugefügt wurde
$role-choice /= „c2pa.redacted”                ; etwas in diesem Bereich wurde redigiert
$role-choice /= "c2pa.subjectArea"             ; Bereich, der für ein Subjekt (menschlich oder nicht) spezifisch
ist
$role-choice /= „c2pa.deleted”                 ; eine Reihe von Informationen wurde entfernt/gelöscht
$role-choice /= „c2pa.styled”                  ; Dieser Bereich wurde stilistisch gestaltet
$role-choice /= „c2pa.watermarked”             ; Dieser Bereich wurde zum Zwecke der weichen Bindung
mit einem Wasserzeichen versehen.
```

18.2.4. Beispiele

Nachfolgend finden Sie eine Reihe von Beispielen in der CBOR-Diagnoseschreibweise (RFC 8949, Abschnitt 8):

```
// Beispiel für einen kombinierten zeitlichen und räumlichen Bereich in einem Video //
{
  „region”: [
    {
      „type”: „temporal”, „time”:
      {
        „type”: „npt”,
        „start“: „0“,
        „end“: „5.2“
      }
    },
    {
      „Typ“: „räumlich“, „Form“:
      {
        „Typ“: „Rechteck“,
        „Einheit“: „Pixel“,
        „Ursprung“: {
           "x": 10.0,
           „y“: 10,0
        },
        „width“: 200.0,
        „height“: 112.0
      }
    }
  ],
  „name“: „Animiertes Logo“, „identifier“:
  „logo-clip“,
  „description“: „5,2 Sekunden des animierten Eröffnungslogos, in einem Rechteck 10 Pixel unterhalb der
Oberkante und 200 Pixel links davon, 200 x 112 Pixel“
}

// Beispiel für einen Textbereich in einer Word-/DOCX-Datei //
{
  „region”: [
    {
      „type”: „textual”, „text” :
      {
        „selectors”: [ [
             {
               „fragment“: „xpointer(/w:document/w:body/w:p/)”
             }
           ]
        ]
      }
    },
  ],
  „description“: „Von KI-Assistent bearbeiteter Inhalt“
}

// Beispiel für einen Textbereich in einer getaggten PDF-Datei //
{
  „region”: [
    {
      „type”: „textual”, „text” :
      {
        „selectors“: [
```

```
            [
              {
                „selector“: {
                  „fragment“: „path=/Document/Sect[0]/P[3]“, „start“: 10,
                  „end“: 20
                }
              }
            ]
          ]
        }
    },
  ],
  „description“: „Redaktion gemäß FOIA-Anfrage durchgeführt“
}

// Beispiel für einen Textbereich in einer nicht getaggten PDF-Datei //
// In diesem Fall können wir nur eine Seite und einen rechteckigen Bereich angeben //
{
  „region“: [
    {
      „type“: „textual“, „text“ :
      {
        „selectors“: [ [
            {
              „selector“: {
                „fragment“: „page=1,rect=10,10,450,500“, „start“: 10,
                „end“: 20
              }
            }
          ]
        ]
      }
    },
  ],
  „Beschreibung“: „Redigierung gemäß FOIA-Anfrage durchgeführt“
}

// Beispiel für das Löschen einiger Seiten aus einer PDF-Datei //
{
  „region“: [
    {
      „type“: „frame“, „frame“ :
      {
        "start" : 27,
        „Ende“: 30
      }
    },
  ],
  „Beschreibung“: „Unnötige Seiten vor der Verteilung entfernt“
}

// Beispiel für einen Zellbereich in Excel //
{
  „region“: [
    {
      „type“: „textual“, „text“:
      {
        „selectors“: [ [
            {
```

```
                „selector“: {
                  „fragment”: „xpointer(Sheet1!A5:A10)”,
                }
              }
            ],
            [
              {
                „selector“: {
                  „fragment”: „xpointer(Sheet1!B5:B10)”,
                }
              }
            ]
          ]
        }
      },
    ],
    „description“: „Einige Formatierungen auf einen Zellbereich in Excel angewendet“
}

// Beispiel für einen zusammenhängenden Bereich von Tabellenzellen //
{
    „region”: [
      {
        „Typ“: „textuell“, „Text“:
        {
          „selectors“: [ [
               {
                 „selector“: {
                   „fragment“: „xpointer(//table[1]/tr[1]/td[2])“,
                 },
                 „end“: {
                   „fragment”: „xpointer(//table[1]/tr[1]/td[4])”,
                 }
               }
            ]
              ]
        }
      },
    ],
    „description“: „Einige Tabellenzellen gelöscht“
}

// Beispiel für einen Bereich einer bestimmten Spur eines Videos //
{
    „region”: [
      {
        „type”: „temporal”, „time”:
        {
          "type": "npt",
          „start“: „0“,
          „end“: „5.2“
        }
      },
      {
        „Typ“: „identifiziert“,
        „Element“: {
          „Bezeichner“: „track_id“, „Wert“: „2“
        }
      }
    ],
   „description“: „Einige Audiospuren wurden verbessert“
```

```
}


// Beispiel für die Angabe, dass die Augen im gesamten Asset geändert wurden //
{
  „region" : [
     {
       „type": „temporal",
       „time": {},
     },
     {
       „Typ": „identifiziert", „Element":
       {
         „identifier": „https://bioportal.bioontology.org/ontologies/FMA", „value": „set of
         eyeballs"
       },
     }
  ]
  „description": „stellte sicher, dass er während des gesamten Meetings so aussah, als würde er schlafen"
}
```

18.3. Metadaten zu Assertions

18.3.1. Beschreibung

In vielen Fällen ist es nützlich oder sogar notwendig, zusätzliche Informationen zu einer Assertion bereitzustellen, z. B. das Datum und die Uhrzeit ihrer Erstellung oder andere Daten, die Manifest-Konsumenten dabei helfen können, fundierte Entscheidungen über die Herkunft oder Richtigkeit der Assertion-Daten zu treffen.

HINWEIS	Ein Manifest-Verbraucher ist nicht verpflichtet, Teile der Metadaten der Assertion zu lesen. Er kann wählen, welche er verwenden möchte, wobei dies sogar je nach Art der Assertion, auf die sie angewendet wird, variieren kann.

Nachfolgend sind die Kernschemata aufgeführt, die in anderen Behauptungen verwendet werden.

Die CDDL-Definition für die Regel „`Assertion-Metadata-Map`" findet sich in CDDL für Assertion-Metadaten:

CDDL für Assertion-Metadaten

```
;Beschreibt zusätzliche Informationen zu einer Assertion, einschließlich einer Hash-URI-Referenz darauf. Wir
verwenden hier einen Socket/Plug, damit die Hash-URI-Zuordnung in einzelnen Dateien verwendet werden kann,
ohne dass die Zuordnung in derselben Datei definiert sein muss
$assertion-metadata-map /= {
  ? „dateTime": tdate, ; Die RFC 3339-Datums- und Zeitzeichenfolge, als die Assertion erstellt/generiert wurde
  ? „reviewRatings": [1* rating-map], ; Bewertungen der Assertion (kann leer sein)
  ? „reference": $hashed-uri-map, ;hashed_uri-Verweis auf eine andere Assertion, auf die sich diese
Überprüfung bezieht
  ? „dataSource": source-map, ; Eine Beschreibung der Quelle der Assertion-Daten, ausgewählt aus einer
vordefinierten Liste
  ? „localizations" : [1* localization-data-entry] ; Lokalisierungen für Zeichenfolgen in der Assertion
  ? „regionOfInterest": $region-map ; beschreibt einen Bereich des Assets, für den diese Assertion relevant
ist
```

```
}

$source-type /= „signer“
$source-type /= „claimGenerator.REE”
$source-type /= „claimGenerator.TEE”
$source-type /= „localProvider.REE”
$source-type /= "localProvider.TEE"
$source-type /= "remoteProvider.1stParty"
$source-type /= „remoteProvider.3rdParty”
$source-type /= „humanEntry”
; Die folgenden beiden Werte von source-type sind ab Version 2.0 veraltet
$source-type /= „humanEntry.anonymous”
$source-type /= „humanEntry.identified”


; HINWEIS: Eine frühere Version dieser Spezifikation enthielt auch ein Feld „actors”, das jedoch in Version 2.0
entfernt wurde.
source-map = {
  „type”: $source-type, ; Ein Wert aus der Aufzählungsliste, der angibt, ob die Quelle der Assertion ein
Claim-Generator ist, der in einer Rich Execution Environment (REE) ausgeführt wird, ein Claim-Generator, der
in einer Trusted Execution Environment (TEE) ausgeführt wird, ein lokaler Datenanbieter in einer REE (z. B.
die Standort-API eines mobilen Betriebssystems), lokale Daten, die in einer TEE ausgeführt werden (z. B. eine
vertrauenswürdige Standort-App eines Chipsatzherstellers), ein Remote-Datenanbieter wie ein Server (z. B. der
Geolokalisierungs-API-Dienst von Google) oder eine Eingabe durch einen Menschen.
  ? „details”: tstr .size (1..max-tstr-length), ; Eine für Menschen lesbare Zeichenfolge mit Details zur
Quelle der Assertion-Daten, z. B. die URL des Remote-Servers, der die Daten bereitgestellt hat
}

int-range = 1..5

$review-code /= „actions.unknownActionsPerformed”
$review-code /= „actions.missing”
$review-code /= „actions.possiblyMissing”
$review-code /= „depthMap.sceneMismatch”
$review-code /= „Zutat.geändert”
$review-code /= „Zutat.möglicherweiseGeändert”
$review-code /= „thumbnail.primaryMismatch”

; Die folgenden drei Werte von review-code sind ab Version 2.0 veraltet
$review-code /= „stds.iptc.location.inaccurate”
$review-code /= „stds.schema-org.CreativeWork.misattributed”
$review-code /= „stds.schema-org.CreativeWork.missingAttribution”

rating-map = {
  „Wert“: int-Bereich, ; „Ein Wert zwischen 1 (schlechteste) und 5 (beste) Bewertung des Artikels“
  ? „code”: $review-code, ; Eine als Label formatierte Zeichenfolge, die den Grund für die Bewertung
beschreibt
  ? „explanation”: tstr .size (1..max-tstr-length), ; Eine für Menschen lesbare Zeichenfolge, die erklärt, warum
die Bewertung so ausfällt, wie sie ausfällt
}

; Die Datenstrukturen, die zum Speichern von Lokalisierungswörterbüchern verwendet werden
$localization-data-entry /= {
  * $$language-string
}

Sprachzeichenfolge /= tstr .size (1..max-tstr-length)
```

Ein Beispiel in CBOR-Diagnoseschrift (RFC 8949, Abschnitt 8):

```
{
  „reference“: {
       "url": "self#jumbf=c2pa.assertions/c2pa.metadata", "alg":
       "sha256",

    },
  „dataSource“: {
    „type“: „localProvider.REE“,
    „details”: „EXIF-GPS-Daten, bereitgestellt von der Geolokalisierungs-API des Betriebssystems”
  }
}
```

In den meisten Fällen erscheinen diese assertionspezifischen Metadaten direkt innerhalb anderer Assertions (z. B. Inhaltsstoffe) als Wert ihres Metadatenfeldes. Manchmal ist es jedoch notwendig oder wünschenswert, die Assertionsmetadaten in einer separaten, unabhängigen Assertionsmetadaten-Assertion zu speichern, beispielsweise wenn eine Assertion nicht im JSON- oder CBOR-Format vorliegt, wie dies bei Miniaturansichten der Fall ist.

Die Bezeichnung für die Metadaten-Assertion lautet `c2pa.assertion.metadata`.

18.3.2. Datenquelle

Dieses Feld „`dataSource`“ ist ein optionales Feld, über das der Anspruchgenerator nachgelagerte Manifest-Verbraucher über die Quelle informieren kann, aus der der Inhalt der Behauptung stammt. Wenn für eine bestimmte Behauptung keine „`dataSource`“ angegeben ist, wird der `Unterzeichner` als „`dataSource`“ betrachtet.

HINWEIS Standardmäßig werden alle `created_assertions` dem Unterzeichner zugeordnet, da das Vertrauensmodell auf dem Vertrauen in den Unterzeichner basiert, der in der Regel auch der Anspruchgenerator ist.

Der Wert des Feldes ist ein dataSource-Objekt, das aus zwei Feldern besteht: `type` und `details`.

Das Feld „dataSource `type`“ definiert den Typ der Datenquelle. Es wird mit Bezeichnungen im Format zusammengestellt, das in Abschnitt 6.2, „Bezeichnungen“, beschrieben ist. Der Wert kann einer der folgenden in Tabelle 5, „Datenquellentypen“, definierten Werte sein, oder es können entitätsspezifische Namespaces als Erweiterungsmechanismus verwendet werden.

Tabelle 5. Datenquellentypen

Wert des `Typs`	**Bedeutung**
`signer`	Der Inhalt der Assertion stammt vom Unterzeichner
`claimGenerator.REE`	Der Inhalt der Behauptung stammt von einem Anspruchgenerator, der in einer Rich Execution Environment (REE) ausgeführt wird, z. B. einem Desktop- oder mobilen Betriebssystem.
`claimGenerator.TEE`	Der Inhalt der Assertion stammt von einem Claim-Generator, der in einer vertrauenswürdigen Ausführungsumgebung (TEE) wie einem vertrauenswürdigen Betriebssystem ausgeführt wird.
`localProvider.REE`	Der Inhalt der Behauptung stammt aus einer Datenquelle, die in einer REE auf demselben physischen Computergerät wie der Anspruchgenerator ausgeführt wird.
Wert vom `Typ`	**Bedeutung**

localProvider.TEE	Der Inhalt der Assertion stammt aus einer Datenquelle, die in einer TEE auf demselben physischen Computergerät wie der Claim-Generator ausgeführt wird.
remoteProvider	Der Inhalt der Assertion stammt aus einer Remote-Datenquelle, die vom Unterzeichner oder vom Anbieter des Claim Generators kontrolliert wird.
remoteProvider.external	Der Inhalt der Assertion stammt aus einer externen, entfernten Datenquelle, die nicht der Unterzeichner oder Anbieter des Anspruchsgenerators ist.
humanEntry	Der Inhalt der Assertion wurde von einem Menschen eingegeben.

Das Feld „Details" ist eine für Menschen lesbare Zeichenfolge, die zusätzliche Informationen über die Datenquelle enthält, z. B. den Namen der API, die zur Bereitstellung der Assertion-Inhalte verwendet wurde, oder die URL des Servers, von dem die Inhalte bereitgestellt wurden. Beispielsweise kann eine allgemeine Standort-Assertion-Quelle den Typwert „remoteProvider.3rdParty" haben, wobei der Wert für „Details" auf „www.googleapis.com/geolocation/v1/geolocate" gesetzt ist.

18.3.3. Bewertungen

Wenn vorhanden, bietet das reviewRatings-Array dem Anspruchgenerator einen Platz, um ein oder mehrere Bewertungsobjekte zur Qualität (oder mangelnden Qualität) einer Assertion bereitzustellen. Ein reviewRatings darf nicht vorhanden sein, wenn ein dataSource-Objekt mit einem type-Feld vorhanden ist, dessen Wert entweder humanEntry.anonymous oder humanEntry.credentialed ist.

Das Wertfeld des Bewertungsobjekts muss einen ganzzahligen Wert zwischen *1* (schlechteste) und *5* (beste) enthalten. Falls vorhanden, muss das Erläuterungsfeld eine für Menschen lesbare Beschreibung der Art der Bewertung enthalten. Zusätzlich kann ein optionales maschinenlesbares Codefeld bereitgestellt werden, das aussagekräftigespezifische Bewertungsergebniscodes definiert. Der Wert des Codefeldes wird unter Verwendung des gleichen Formats definiert, das in Abschnitt 6.2, „Labels", beschrieben ist. Der Wert kann einer der folgenden in Tabelle 6, „Werte des Codefeldes", definierten Werte sein, oder es können entitätsspezifische Namespaces als Erweiterungsmechanismus verwendet werden.

Tabelle 6. Werte des Codefelds

Wert des Codes	**Anwendbare Assertion**	**Bedeutung**
actions.unknownActionsPerformed	c2pa.actions	Die Aktionsaussage enthält keine vollständige Liste aller im Authoring-Tool durchgeführten Aktionen (z. B. aufgrund der Verwendung eines Filters eines Drittanbieters, dessen Wirkung dem Authoring-Tool unbekannt ist).
actions.placedIngredientNotFound	c2pa.actions	Die überprüfte Aktionsaussage enthält eine platzierte Aktion ohne eine auflösbare Zutaten-URI. Der Wert sollte 1 sein.
Zutat.Aktion fehlt	c2pa.ingredient	Die überprüfte Zutat-Aussage enthält nicht mindestens eine Aktion, die in ihrer Angabe darauf verweist. Der Wert sollte 1 sein.
Zutat.nicht sichtbar	c2pa.Zutat	Die zu prüfende Inhaltsstoffangabe ist in den mit diesem Manifest verbundenen digitalen Inhalten nicht sichtbar. Der Wert sollte 1 sein.
Wert des Codes	**Anwendbare Angabe**	**Bedeutung**
depthMap.sceneMisÜbereinstimmung	c2pa.depthmap.GDe pth	Der Inhalt der Tiefenkarten-Assertion entspricht nicht der Szene, die in der primären Darstellung des Assets dargestellt wird (z. B. aufgrund eines Bild-in-Bild-Angriffs).

`thumbnail.primary Nicht übereinstimmend`	`c2pa.thumbnail.cl aim`	Der Inhalt der Miniaturansicht stimmt nicht mit dem Inhalt der primären Darstellung im Asset überein.

18.3.4. Referenzen

Da das Referenzfeld der Assertion-Metadaten-Assertion ein Standard-hashed_uri ist, ist es auch möglich, dass eine Assertion-Metadaten-Assertion auf Assertions in anderen Manifesten als dem aktiven Manifest verweist. Das aktive Manifest kann beispielsweise eine Assertion-Metadaten-Assertion enthalten, die die im Manifest eines Inhaltsstoffs vorhandene `c2pa.` metadata-Assertion validiert.

HINWEIS Da es sich bei der Behauptung um eine spezielle Art von Assertion handelt, kann dieselbe Methode verwendet werden, um auf Behauptungen in anderen Manifesten zu verweisen.

18.3.5. DateTime

Wenn ein dateTime-Feld vorhanden ist, muss sein Wert eine Datums- und Uhrzeitzeichenfolge sein, die den CBOR-Datums-/Uhrzeitangaben (RFC 8949, 3.4.1) entspricht.

18.3.6. Region von Interesse

Die Angabe kann sich nur auf einen Teil eines Assets beziehen, z. B. auf einen Bereich von Frames in einem Video oder einen bestimmten Bereich eines Bildes. Ein solcher Teil kann mithilfe eines Feldes „`regionOfInterest`“ identifiziert werden, dessen Wert ein Region-Map-Objekt ist (wie in Abschnitt 18.2, „Regions of Interest“, definiert).

18.3.7. Lokalisierung

18.3.7.1. Allgemeines

Es ist wichtig, dass die Nutzer von C2PA-Manifesten die Informationen nach Möglichkeit in ihrer Muttersprache verstehen können. Zu diesem Zweck ist es möglich, Lokalisierungsinformationen für eine Behauptung mit einem Wörterbuch hinzuzufügen, das in den Metadaten der Behauptung enthalten ist.

18.3.7.2. Lokalisierungswörterbuch

Ein Lokalisierungswörterbuch besteht aus einem einzigen Objekt, wobei jeder seiner Schlüssel die Übersetzungen unter Verwendung der Sprachindexierungstechnik aus JSON-LD darstellt. Wenn der zu übersetzende Wert nicht mit einem Schlüssel der obersten Ebene verknüpft ist, muss die „Punktnotation” (`.`) verwendet werden, um auf in Objekten verschachtelte Schlüssel zu verweisen. Die Array-Indexierungsnotation (`[n]`, `n>=0`) muss verwendet werden, wenn ein bestimmtes Element in einem Array durchlaufen werden muss. Wenn der zu übersetzende Wert selbst ein Array ist, kann auf ein bestimmtes Element verwiesen werden. Einige Beispiele sind in Beispiel 4, „Beispiele für Lokalisierungswörterbücher

Wörterbücher” aufgeführt:

Beispiel 4. Beispiele für Lokalisierungswörterbücher

```
{
  „dc:title“: {
     "en-US": "Kevins fünf Katzen",
     „en-GB“: „Lord Kevins fünf Katzen“, „es-MX“:
     „Los Cinco Gatos de Kevin“, „es-ES“: „Los
     Thinco Gatos de Kevin“, „fr“: „Les Cinq Chats
     de Kevin“,
     „jp“: „ケヴィンの５匹の猫“
  }
}
```

```
{
  „actions[0].softwareAgent”: { „en-US”:
     „Joe's Photo Editor”, „en-GB”: „Joe's
     Photo Editor”,
     "es": "Joe's Fotoeditor", "fr": "Joe's
     Fotoeditor",
     „jp“: „ジョーの写真編集者“
  }
}
```

Alle derartigen Schlüssel oder Werte von Drittanbietern müssen gemäß Abschnitt 6.2.1, „Namensräume“, mit einem Namensraum versehen werden.

z. B. `com.litware`. Damit ein Manifest-Consumer für Menschen lesbare Informationen zu diesen Schlüsseln und Werten anzeigen kann, sollte der Anspruchgenerator die Zeichenfolgen über diesen Lokalisierungsansatz bereitstellen.

Lokalisierte Aktionen zeigen ihre Verwendung bei der Lokalisierung benutzerdefinierter Aktionen, indem sie in den Assertion-Metadaten eines `c2pa.`actions-Assertion.

Lokalisierte Aktionen

```
{
  „com.litware.blur”: { „en-
     US”: „Blur”,
     „fr-FR“: „Brouiller“,
  },
  „com.litware.filter”: { „en-
     US”: „Filter”,
     "es-ES": "Filtrar",
     „jp-JP“: „フィルター“
  }
}
```

18.4. Zusammenfassung der Standard-C2PA-Assertions

Die Standard-C2PA-Aussagen sind in Tabelle 7, „Standard-C2PA-Aussagen“, aufgeführt:

Tabelle 7. Standard-C2PA-Assertions

Typ	Assertion	Schema	Serialisierung
Aktionen	c2pa.actions, c2pa.actions.v2	C2PA	CBOR
Assertion-Metadaten	c2pa.assertion.metadata	C2PA	CBOR
Asset-Referenz	c2pa.asset-ref	C2PA	CBOR
Asset-Typ	c2pa.asset-type (veraltet), c2pa.asset-type.v2	C2PA	CBOR
BMFF-basierter Hash	c2pa.hash.bmff (entfernt), c2pa.hash.bmff.v2 (veraltet), c2pa.hash.bmff.v3	C2PA	CBOR
Zertifikatsstatus	c2pa.certificate-status	C2PA	CBOR
Cloud-Daten	c2pa.cloud-data	C2PA	CBOR
Sammlungsdaten-Hash	c2pa.hash.collection.data	C2PA	CBOR
Daten-Hash	c2pa.hash.data	C2PA	CBOR
Tiefenkarte	c2pa.depthmap.GDepth	https://developers.google.com/depthmap-metadata/reference	CBOR
Eingebettete Daten	c2pa.embedded-data	C2PA	JUMBF Eingebettete Datei
Schriftartinformationen	font.info	C2PA	CBOR
Allgemeiner Box-Hash	c2pa.hash.boxes	C2PA	CBOR
Zutat	c2pa.Zutat, c2pa.Zutat.v2, c2pa.Zutat.v3	C2PA	JUMBF-eingebettete Datei
Metadaten	c2pa.metadata	C2PA	JSON-LD
Multi-Asset-Hash	c2pa.hash.multi-asset	C2PA	CBOR
Soft Binding	c2pa.soft-binding	C2PA	CBOR
Miniaturansicht	c2pa.thumbnail.claim (Zeitpunkt der Erstellung des Anspruchs), c2pa.thumbnail.ingredient (Import einer Zutat)	C2PA	Eingebettete Datei
Zeitstempel	c2pa.time-stamp	C2PA	CBOR

18.5. Daten-Hash

18.5.1. Beschreibung

Die gängigste Methode zur eindeutigen Überprüfung der Integrität von Teilen eines nicht BMFF-basierten Assets ist die Verwendung von Hard Bindings (d. h. kryptografischen Hashes) in Daten-Hash-Assertions. Für Formate, die „boxähnlich" sind, aber nicht mit BMFF kompatibel sind, wird jedoch die allgemeine Box-Hash-Assertion empfohlen.

Die Daten-Hash-Assertion unterstützt die Erstellung und Speicherung von Hashes, wie in Abschnitt 13.1, „Hashing", beschrieben, und der Wert muss im Hash-Feld vorhanden sein.

Jede Daten-Hash-Aussage definiert einen bestimmten Byte-Bereich, über den der Hash berechnet wurde. Wenn nur ein Teil des Assets gehasht werden soll, müssen die auszuschließenden Bereiche im Array-Wert des Feldes `„Exclusions"` (Ausschlüsse) angegeben werden. Diese ausgeschlossenen Bereiche müssen nach aufsteigender Startposition geordnet sein und dürfen sich nicht überschneiden.

Bei Daten-Hash-Ausschlussbereichen muss der Bereich innerhalb derselben logischen Einheit (z. B. Box, Segment, Objekt) beginnen und enden und darf sich nicht mit einem Header- oder Längenfeld überschneiden, das mit dieser Einheit verbunden ist, mit Ausnahme von Freebox- oder Pad-Daten. Es liegt in der Verantwortung des Anspruchsgenerators, Ausschlussbereiche so zu definieren, dass sichergestellt ist, dass Daten, die ein Angreifer in diesen Bereichen platzieren könnte, die Interpretation des Assets nicht wesentlich beeinflussen können. Darüber hinaus muss der Anspruchgenerator sicherstellen, dass der Ausschlussbereich nur Inhalte aus dem C2PA Manifest Store oder Metadaten der Ressource (z. B. EXIF-, IPTC-Metadaten) enthält. Beispiele für Metadaten, die übersprungen werden könnten, sind nicht verifizierte Benutzernamen oder Informationen zur Bilddrehung.

Eine frühere Version dieser Spezifikation enthielt ein URL-Feld, um einen Verweis auf den Speicherort der Hash-Daten anzugeben, das jedoch nie verwendet wurde. Dieses Feld ist nun zugunsten der Asset-Referenz-Assertion veraltet. Claim-Generatoren dürfen dieses Feld nicht zu einer Daten-Hash-Assertion hinzufügen, und Verbraucher müssen das Feld ignorieren, wenn es vorhanden ist, außer dass dies keinen Einfluss auf die Einbeziehung des Feldes als Teil des zu validierenden Inhalts hat, wie in Abschnitt 15.10.3, „Assertion Validation" (Assertion-Validierung), beschrieben.

Eine Daten-Hash-Assertion muss die Bezeichnung `c2pa.hash.data` haben.

Eine Daten-Hash-Assertion darf nicht in einer Cloud-Daten-Assertion erscheinen.

Eine Daten-Hash-Assertion darf nicht mit einem komprimierten Manifest verwendet werden.

HINWEIS Diese Einschränkung besteht aufgrund einer technischen Inkompatibilität zwischen den beiden.

18.5.2. Schema und Beispiel

Das Schema für diesen Typ wird durch die data-hash-map-Regel in der folgenden CDDL-Definition definiert:

```
; Überprüfen Sie auch die Optionalität innerhalb der Hash-Map
; Die Datenstruktur, die zum Speichern des kryptografischen Hashs einiger oder aller Daten des Assets verwendet
wird

; und zusätzliche Informationen, die zur Berechnung des Hashs
erforderlich sind. data-hash-map = {
   ? „exclusions": [1* EXCLUSION_RANGE-map], ; Bereiche haben monoton steigende „start"-Werte, und keine zwei
Bereiche dürfen sich überschneiden.
   ? „alg":tstr .size (1..max-tstr-length), ; Eine Zeichenfolge, die den kryptografischen Hash-Algorithmus
identifiziert, der zur Berechnung des Hash in dieser Assertion verwendet wird, entnommen aus der C2PA-Hash-
Algorithmus-Identifikatorliste. Fehlt dieses Feld, wird der Hash-Algorithmus als Wert „alg" der
```

```
umgebenden Struktur übernommen. Sind beide vorhanden, wird das Feld in dieser Struktur verwendet. Ist an
keiner dieser Stellen ein Wert vorhanden, ist diese Struktur ungültig; es gibt keinen Standardwert.
  „hash”: bstr, ; Byte-Zeichenkette des Hash-Werts
  „pad”: bstr, ; mit Nullen aufgefüllte Byte-Zeichenkette zum Auffüllen von Speicherplatz
  ? „pad2”: bstr, ; optionale mit Nullen gefüllte Byte-Zeichenkette zum Auffüllen von Speicherplatz
  ? „name“: tstr .size (1..max-tstr-length), ; (optional) eine für Menschen lesbare Beschreibung dessen, was
dieser Hash abdeckt
  ? „url”: uri, ; Nicht verwendet und veraltet.
}

EXCLUSION_RANGE-map = {
  „start”: uint, ; Startbyte des Bereichs „length”: uint, ;
  Anzahl der zu ausschließenden Datenbytes
}
```

Ein Beispiel in CBOR-Diagnoseschrift (RFC 8949, Abschnitt 8) ist unten aufgeführt:

```
{
  „alg” : „sha256”,
  „pad”: '0000',
  "hash": 'Auxjtmax46cC2N3Y9aFmBO9Jfay8LEwJWzBUtZ0sUM8gA=', "name": "JUMBF
  manifest"
  "exclusions": [
    {
      „start“: 9960,
      „length”: 4213
    }
  ],
}
```

Normalerweise werden die `Start`- und Längenwerte einer `Ausschließung` in ihrer bevorzugten Serialisierung (d. h. „so kurz wie möglich“) geschrieben. Wenn jedoch eine Daten-Hash-Assertion erstellt werden muss, die `Start`- und Längenwerte jedoch noch nicht bekannt sind, werden sie „so groß wie möglich“ erstellt, d. h. als 32-Bit-Ganzzahl.

`Der` Pad-Wert muss immer vorhanden sein, sollte jedoch eine mit Nullen aufgefüllte Byte-Zeichenkette der Länge 0 sein, sofern er nicht zum Ersetzen (d. h. „Auffüllen”) von Bytes während der Mehrfachdurchlaufverarbeitung verwendet wird. `pad2` ist eine optionale mit Nullen aufgefüllte Byte-Zeichenkette, die verwendet wird, wenn die gewünschte Auffüllung mit `pad` nicht erreicht werden kann.

HINWEIS In Abschnitt 10.4, „Mehrstufige Verarbeitung”, wird beschrieben, wie die richtigen Werte eingegeben und die Auffüllung angepasst werden.

18.5.3. Besondere Überlegungen für JPEG 1

Beim Hashing einer JPEG 1-Datei (.jpg), in die das C2PA-Manifest eingebettet wird, müssen der APP11-Marker (`FFEB`) und die Segmentlänge (`Lp`) aller APP11-Segmente, die die JUMBF-Daten enthalten, in den Ausschlussbereich aufgenommen werden.

HINWEIS Alle APP11-Segmente, die das C2PA-Manifest JUMBF enthalten, sind zusammenhängend, sodass nur ein einziger Bereich erforderlich ist.

18.5.4. Besondere Überlegungen für PNG

Beim Hashing einer PNG-Datei (.png), in die das C2PA-Manifest eingebettet wird, ist es wichtig, dass die `Länge` und der „`caBX`" (der den Chunk-Typ angibt) des Chunks, der die JUMBF-Daten enthält, in den Ausschlussbereich aufgenommen werden.

18.6. BMFF-basierter Hash

18.6.1. Beschreibung

Teile eines BMFF-basierten Assets, die ein Anspruchsgenerator eindeutig mit einer festen Bindung (d. h. einem kryptografischen Hash) identifizieren möchte, müssen mithilfe von BMFF-basierten Hash-Assertions beschrieben werden.

Eine BMFF-basierte Hash-Assertion muss die Bezeichnung `c2pa.hash.bmff.v3` tragen.

HINWEIS	Frühere Versionen dieses Standards dokumentierten auch `c2pa.hash.bmff`- und `c2pa.hash.bmff.v2` Assertions dokumentiert.

WICHTIG	Validatoren müssen alle `c2pa.hash.bmff`-Assertions ignorieren und das Manifest so verarbeiten, als ob die Assertion nicht vorhanden wäre.

Eine BMFF-basierte Hash-Assertion darf nicht in einer Cloud-Daten-Assertion erscheinen.

Eine frühere Version dieser Spezifikation enthielt ein URL-Feld, um einen Verweis auf den Speicherort der Hash-Daten anzugeben, das jedoch nie verwendet wurde. Dieses Feld ist nun zugunsten der Asset-Referenz-Assertion veraltet. Claim-Generatoren dürfen dieses Feld nicht zu einer BMFF-Hash-Assertion hinzufügen, und Verbraucher müssen das Feld ignorieren, wenn es vorhanden ist, außer dass dies keinen Einfluss auf die Einbeziehung des Feldes als Teil des zu validierenden Inhalts hat, wie in Abschnitt 15.10.3, „Assertion Validation" (Assertion-Validierung), beschrieben.

18.6.2. Hash-Berechnung

Um den im Wertfeld eines BMFF-Hash angegebenen Hash zu berechnen, werden alle Bytes der Datei zum Hash hinzugefügt, mit Ausnahme derjenigen BMFF-Boxen oder Teilmengen davon, die mit einem Ausschlusseintrag im Ausschlussarray übereinstimmen.

Boxen, die vollständig enthalten sind, enthalten auch ihre Box-Header in den Eingabedaten, die zum Hash beitragen. Ebenso werden bei Boxen, die vollständig ausgeschlossen sind, auch ihre Box-Header aus den Eingabedaten ausgeschlossen, die zum Hash beitragen. Wenn ein Feld durch die Verwendung eines Teilfeldes in der Ausschlussspezifikation teilweise aus den Eingabedaten ausgeschlossen wird, die zum Hash beitragen, sind die durch die relativen Byte-Offsets im Teilfeld definierten Teile des auszuschließenden Feldes Offsets vom Anfang des Feldes einschließlich der Feldüberschriften und keine Offsets vom Anfang des Feldinhalts. Diese Teilbereiche müssen nach steigendem Offsetwert geordnet sein und dürfen sich nicht überschneiden.

In einer `c2pa.hash.bmff.v2` (veraltet) und `c2pa.hash.bmff.`v3-Assertion bestehen die Eingabedaten, die zum Hash für diese Root-Box beitragen, aus der Verkettung der Binärzeichenfolgen `offset || data`, wobei `offset` als der absolute Datei-Offset der Box als 8-Byte-Ganzzahl im Big-Endian-Format definiert ist und `data` als der Inhalt der Box einschließlich der Header abzüglich etwaiger Ausschlüsse definiert ist. In dieser Definition steht „||"

die binäre Verkettung der beiden. Der Offset darf nicht in Merkle-Tree-Hashes enthalten sein, wenn die bmff-hash-map sowohl das `Hash-` als auch das Merkle-Feld enthält.

Darüber hinaus enthalten die Assertions `c2pa.hash.bmff.v2` (veraltet) und `c2pa.hash.bmff.v3` die folgenden Funktionen:

- Der absolute Dateibyte-Offset ist am Anfang der Eingabedaten enthalten, die zum Hash für jede Root-Box beitragen. Dadurch wird sichergestellt, dass eine im Hash enthaltene Root-Box ihre Position in der Datei nicht ändern kann.
- Die mdat-Box wird nicht mehr vollständig ausgeschlossen, wenn die bmff-hash-map sowohl die `Hash-` als auch `die Merkle-Felder` enthält enthält. Stattdessen wird durch einen obligatorischen Eintrag in der Ausschlussliste der größte Teil der Box ausgeschlossen.

HINWEIS

Diese beiden Funktionen stellen sicher, dass sich die Position der `MDAT-Datei` in der Datei nicht ändern kann, und verhindern gleichzeitig
die Notwendigkeit des Offsets für jeden einzelnen Merkle-Tree-Hash, wenn die bmff-hash-map sowohl die `Hash-` als auch die Merkle-Felder enthält.

Ein Feld stimmt genau dann mit einem Ausschlusseintrag im Ausschlussarray überein, wenn alle folgenden Bedingungen erfüllt sind:

- Die Position der Box in der Datei entspricht dem XPath-Feld des Eintrags `in der Ausschlusskarte`. Beispielsweise würde der Ausschluss-XPath
  `/foo/bar[2]` würde mit den Positionen `/foo[3]/bar[2]` und `/foo[2]/bar[2]`, aber nicht `/foo[3]/bar[1]` oder `/foo[3]/bar[2]/baz[1]`.
- Wenn `die Länge` im Eintrag `der Ausschlusszuordnung` angegeben ist, entspricht die Länge der Box genau dem `Längenfeld des Eintrags in der Ausschlusszuordnung`
  des Eintrags in der Ausschlusszuordnung übereinstimmen. Hinweis: Die Länge umfasst die Box-Header.
- Wenn `die Version` im Eintrag `der Ausschlusszuordnung` angegeben ist, handelt es sich um eine FullBox, und die Version der Box stimmt genau mit dem Versionsfeld des Eintrags in `der Ausschlusszuordnung` überein.
- Wenn `flags` (Byte-Array mit genau 3 Bytes) im Eintrag `der Ausschlusszuordnung` angegeben ist und die Box eine FullBox ist. Wenn `exact` auf true gesetzt oder nicht angegeben ist, stimmen die Flags der Box (Bit(24), d. h. 3 Bytes) ebenfalls genau mit dem Feld `flags` des Eintrags in `der Ausschlusszuordnung` überein. Wenn `„exact“` auf „false“ gesetzt ist, stimmt das Bitweise-Und der Flags der Box (Bit(24), d. h. 3 Bytes) mit dem Flags-Feld des Eintrags in `der Ausschlusszuordnung` genau mit dem Flags-Feld des Eintrags in der Ausschlusszuordnung überein (d. h., die Box hat mindestens diese Bits gesetzt, kann aber auch zusätzliche Bits gesetzt haben).
- Wenn `Daten` (Array von Objekten) im Eintrag `der Ausschlusszuordnung` angegeben sind, dann stimmen für jedes Element im Array die Binärdaten der Box im relativen Byte-Offset-Feld dieses Elements genau mit dem Byte-Feld dieses Elements überein.

Die Zeichenkettensyntax des `xpath`-Feldes ist auf die folgende strenge Teilmenge beschränkt.

- Es darf nur die abgekürzte Syntax verwendet werden.
- Es dürfen nur vollständige Pfade verwendet werden.
- Es darf nur die Knotenauswahl über `node` oder `node[integer]` verwendet werden.
- Die Nachkommenssyntax, d. h. `//`, darf NICHT verwendet werden.
- Alle Knoten müssen BMFF`-4cc`-Codes sein.

Beispiel 5. Vollständige Syntax für das Feld „xpath“

```
   xpath = '/' nodes nodes =
   node
          | node '/' nodes node =
   box4cc
         | box4cc '[' integer ']'
Wobei:
   box4cc ist eine beliebige 4cc, die gemäß ISO/IEC 14496-12 für eine BMFF-Box
   zulässig ist. integer ist eine beliebige positive ganze Zahl ungleich Null ohne
   führende Nullen.
```

Jeder Ausschlusseintrag kann mit null oder mehr Feldern übereinstimmen. Es ist nicht erforderlich, dass ein Ausschlusseintrag genau mit einem Feld übereinstimmt.

Ein Nicht-Blatt-XPath-Knoten darf nur auf ein Containerfeld verweisen, das keine eigenen Felder hat (d. h. keine Daten enthält, sondern nur untergeordnete Felder) und nicht von FullBox erbt. Dadurch wird sichergestellt, dass ein C2PA-Validator die Syntax und Semantik ungewöhnlicher Felder, die andere Felder enthalten, nicht kennen muss. Wenn ein untergeordnetes Feld einer solchen ungewöhnlichen Box ganz oder teilweise ausgeschlossen werden muss, muss das XPath-Feld des `Ausschluss-Map`-Eintrags auf die ungewöhnliche Box selbst verweisen und das Subset-Map-Feld muss den/die Bytebereich(e) ausschließen, der/die die ausgeschlossenen Daten des untergeordneten Feldes enthält/enthalten. Beispielsweise enthält die Box `„sgpd“` andere Boxen, ist jedoch insofern ungewöhnlich, als sie von FullBox erbt. Wenn also untergeordnete Boxen ganz oder teilweise aus `„sgpd“` ausgeschlossen werden müssen, muss die Assertion ein xpath-Feld verwenden, das auf `„sgpd“` selbst verweist (z. B.
`/moof/traf/sgpd`) und verwendet das Feld `„subset-map“,` um die gewünschten Bytes auszuschließen.

Wenn das C2PA-Manifest in die Datei eingebettet ist, muss das Feld, das es enthält, einer der Einträge im `Ausschlussarray` sein
. Weitere Informationen finden Sie in Abschnitt A.5, „Einbetten von Manifesten in BMFF-basierte Assets“.

Wenn eine nicht root-ausgeschlossene Box nach der Erstellung des C2PA-Manifests entfernt wird, muss sie durch eine `„freie”` Box derselben Größe ersetzt werden, um sicherzustellen, dass die Eingabedaten, die zum Hash für andere Boxen beigetragen haben, nicht ungültig werden. Wenn die Speichergröße des C2PA-Manifests nach dessen Erstellung durch Verwendung eines komprimierten Manifests reduziert wird, muss an seiner Stelle eine `„freie”` Box eingefügt werden, um sicherzustellen, dass die Offsets gleich bleiben. Wenn zu erwarten ist, dass nach der Erstellung des C2PA-Manifests ein nicht-root-ausgeschlossener Bereich hinzugefügt wird, muss zum Zeitpunkt der Manifest-Erstellung ein `„freier”` Bereich mit ausreichend Platz für den ausgeschlossenen Bereich eingefügt werden, und dieser `„freie”` Bereich muss ebenfalls durch einen Ausschlusseintrag unter Verwendung seines vollständigen XPaths ausgeschlossen werden. Wenn das ausgeschlossene Feld hinzugefügt oder die Speichergröße des C2PA-Manifests erhöht wird, muss das `„freie”` Feld verkleinert (oder entfernt) werden, um die hinzugefügten Daten auszugleichen. Wenn jedoch nicht genügend Platz im `„freien”` Feld vorhanden ist, muss ein Standardmanifest verwendet werden.

Das Einbetten von C2PA-Daten in eine BMFF-basierte Ressource über MP4-Boxen verändert die Datei-Offsets in anderen MP4-Boxen sowie die absoluten Datei-Byte-Offsets, die in den Eingabedaten enthalten sind, die zum Hash für jede Root-Box beitragen. Diese Boxen und Offsets müssen mit ihren Werten nach dem Einbetten und nicht mit ihren Werten vor dem Einbetten in die Eingabedaten aufgenommen werden, die zum Hash beitragen, da sonst die BMFF-basierte Hash-Assertion nicht validiert wird.

Es gibt drei Möglichkeiten, wie eine Implementierung sicherstellen kann, dass die Werte nach der Einbettung für alle Dateibyte-Offsets gehasht werden:

1. Verwenden Sie `„freie”` Boxen.

a. Bestimmen Sie die angemessene(n) maximale(n) Größe(n) für die C2PA-Box(en), die eingebettet werden sollen. Alle MP4-Boxen für C2PA unterstützen ungenutzte Füllbytes am Ende, sodass es in Ordnung ist, die Größe für die „freien" Boxen zu überschätzen, da alle zusätzlichen Bytes ignoriert werden.

b. Fügen Sie „freie" Box(en) der angegebenen Größe(n) in die Asset-Datei(en) ein und aktualisieren Sie alle Offsets entsprechend.

c. Führen Sie ein Hashing des Assets mit „/free" auf der Ausschlussliste durch.

d. Erstellen und signieren Sie das Manifest. Erstellen Sie die C2PA-Box(en).

e. Überschreiben Sie die „freien" Box(en) mit den C2PA-Box(en).

2. Verwenden Sie einen Zwei-Durchlauf-Ansatz.

a. Berechnen Sie die genauen Größen der BMFF-basierten Hash-Assertion und der Merkle-Box(en), falls vorhanden. Letzteres erfordert das Parsen der Asset-Datei(en), um die Größe des Merkle-Baums zu bestimmen.

b. Berechnen Sie die genaue Größe des endgültigen Manifests.

c. Führen Sie ein Hashing der Asset-Datei(en) durch. Aktualisieren Sie alle Boxen, die Datei-Offsets enthalten, auf korrekte Werte, bevor Sie diese Box in die Eingabedaten aufnehmen, die zum Hash beitragen. Berechnen Sie die Eingabedaten, die zum Hash beitragen, unter Verwendung von (Offset || Daten) unter Verwendung des aktualisierten absoluten Datei-Offsets, wie oben beschrieben. Wie oben angegeben, ist der Offset nicht in den Daten enthalten, die zu Merkle-Tree-Hashes beitragen, wenn die bmff-hash-map sowohl das Hash- als auch das Merkle-Feld enthält.

d. Erstellen und signieren Sie das Manifest. Erstellen Sie die C2PA-Box(en).

e. Fügen Sie die C2PA-Box(en) ein.

3. Aktualisierten Manifest-Speicher am Ende der BMFF-Datei platzieren.

a. Setzen Sie den ursprünglichen Manifest-Speicher box_purpose von Manifest auf Original.

b. Erstellen und signieren Sie das Manifest.

c. Erstellen Sie eine C2PA ContentProvenanceBox mit box_purpose auf „update".

d. Fügen Sie „updatedManifest" in C2PA ContentProvenanceBox ein.

e. Fügen Sie die C2PA ContentProvenanceBox am Ende der BMFF-Datei ein.

f. Wenn ein Standardmanifest hinzugefügt wird, während ein Update-Manifestspeicher vorhanden ist, wird der Inhalt des Update-Manifestspeichers in das „Original"-Manifest verschoben.

g. Der aktualisierte Manifest-Speicher wird dann vom Ende der Datei entfernt, wodurch die Abwärtskompatibilität mit einem einzigen Manifest für gängige Anwendungsfälle gewährleistet ist.

h. Der „ursprüngliche" Manifest-Speicher box_purpose wird wieder in manifest geändert und das Standard-Manifest wird wie gewohnt hinzugefügt.

HINWEIS

Das Feld „box_purpose" ist nicht im Hash enthalten und kann geändert werden, ohne dass dadurch die Gültigkeit beeinträchtigt wird.

bestehenden Hash ungültig. Ebenso führt das Anhängen des neuen C2PA ContentProvenanceBox nicht zur Ungültigkeit bestehender Hashes.

Die Zwei-Durchlauf-Methode ist zwar deutlich komplexer, ermöglicht jedoch eine korrekte Hash-Berechnung ohne

Vorabkenntnis der maximalen Manifestgröße. Außerdem wird die Größe der endgültigen Datei minimiert. Zu den gängigen Boxen (**nicht** vollständig) mit Datei-Offsets gehören „iloc", „stco", „co64", „tfhd", „sidx" und „saio".

Die Option, aktualisierte Manifeste am Ende der BMFF-Datei zu platzieren, ermöglicht Aktualisierungen, wenn kein ausreichend großer „freier" Block vorhanden ist oder wenn die Komplexität des Zwei-Durchlauf-Ansatzes nicht erwünscht ist. Diese Option unterstützt auch Chunk-Offsets in Atom-„stco"-Blöcken mit partiellen Daten-Offset-Informationen.

18.6.3. Schema und Beispiel

Das Schema für die (veralteten) Assertions `c2pa.hash.bmff.v2` und `c2pa.hash.bmff.v3` wird durch die `bmff-hash-map`-Regel in der folgenden CDDL-Definition definiert:

```
bmff-hash-map = {
  „exclusions": [1* exclusions-map],
  ? „alg": tstr .size (1..max-tstr-length), ; Eine Zeichenfolge, die den zur Berechnung dieses Hashwerts
verwendeten kryptografischen Hashalgorithmus identifiziert und aus der C2PA-Hashalgorithmus-Identifikatorliste
stammt. Wenn dieses Feld fehlt, wird der Hash-Algorithmus aus einer umgebenden Struktur übernommen, wie durch
diese Struktur definiert. Wenn beide vorhanden sind, wird das Feld in dieser Struktur verwendet. Wenn an keiner
dieser Stellen ein Wert vorhanden ist, ist diese Struktur ungültig; es gibt keinen Standardwert.
  ? „hash": bstr, ; Bei nicht fragmentierten MP4-Dateien ist dies der Hash der gesamten BMFF-Datei mit
Ausnahme der im Ausschlussarray aufgeführten Boxen. Bei fragmentierten MP4-Dateien darf dieses Feld nicht
vorhanden sein.
  ? „merkle": [1* merkle-map], ; Eine Reihe von Merkle-Baumzeilen und die zugehörigen Daten, die erforderlich
sind, um die Überprüfung einer einzelnen „mdat"-Box, mehrerer „mdat"-Boxen und/oder einzelner Fragmentdateien
innerhalb des Assets zu ermöglichen.
  ? „name": tstr .size (1..max-tstr-length), ; optional) eine für Menschen lesbare Beschreibung dessen, was
dieser Hash abdeckt.
  ? „url": uri, ; Unbenutzt und veraltet.
}

;(optional) CBOR-Byte-Zeichenkette mit genau 3 Bytes.
flag-type = bytes

flag-t = flag-type .eq 3

exclusions-map = {
  „xpath": tstr, ; Position der Box(en), die aus dem Hash ausgeschlossen werden sollen, beginnend mit dem
Stammknoten als xpath-formatierte Zeichenfolge der Version https://www.w3.org/TR/xpath-10/ mit stark
eingeschränkter Syntax.
  ? „length": uint, ; (optional) Länge, die eine Box am äußersten Rand haben muss, um aus dem Hash
ausgeschlossen zu werden.
  ? „data": [1* data-map], ; (optional) Die Daten im untersten Feld am angegebenen relativen Byte-Offset
müssen mit den angegebenen Daten identisch sein, damit das Feld aus dem Hash ausgeschlossen wird.
  ? „subset":[1* subset-map], ; (optional) Nur dieser Teil des ausgeschlossenen Feldes wird aus dem Hash
ausgeschlossen. Jeder Eintrag im Array muss einen monoton steigenden relativen Byte-Offset haben. Keine
Teilmenge innerhalb des Arrays darf sich überschneiden. Der letzte Eintrag kann eine Länge von Null haben;
dies bedeutet, dass der Rest der Box ab diesem relativen Byte-Offset ausgeschlossen ist. Ein relativer
Byte-Offset oder ein relativer Byte-Offset plus Länge, der die Länge der Box überschreitet, ist zulässig; Bytes
jenseits des Endes der Box werden niemals gehasht.
  ? „version": int, ; (optional) Version, die in einer Leafmost-Box festgelegt werden muss, damit die Box aus
dem Hash ausgeschlossen wird. Nur für eine Box angegeben, die von FullBox erbt.
  ? „flags": flag-t, ; (optional) Byte-Zeichenkette mit genau 3 Bytes. Die 24-Bit-Flags, die in einer
Leafmost-Box gesetzt sein müssen, damit die Box aus dem Hash ausgeschlossen wird. Nur für eine Box angegeben,
die von FullBox erbt.
  ? „exact": bool, ; (optional) gibt an, dass die Flags exakt übereinstimmen müssen. Wenn nicht angegeben, ist
der Standardwert „true". Wird nur für eine Box angegeben, die von FullBox erbt, und wenn
```

```
flags ebenfalls angegeben ist.
}

data-map = { "offset":
   uint, "value" : bstr,
}
subset-map = {
   "offset": uint,
   "length": uint,
}

; Jeder Eintrag in einer Zuordnung ist eine Merkle-Baumzeile und die zugehörigen Daten, die zur Validierung
einer einzelnen
; „mdat“-Box oder mehrere „mdat“-Boxen innerhalb des Assets.",
merkle-map = {
   „uniqueId”: int, ; 1-basierte eindeutige ID, die zur Unterscheidung zwischen Dateien verwendet wird, um zu
bestimmen, welcher Merkle-Baum zur Validierung einer bestimmten „mdat”-Box verwendet werden soll.
   „localId”: int, ; Eine lokale ID, die den Merkle-Baum angibt.
   „count”: int, ; Anzahl der Blattknoten im Merkle-Baum. Nullknoten werden in dieser Zählung nicht
berücksichtigt.
   ? „alg”: tstr .size (1..max-tstr-length), ; Eine Zeichenfolge, die den kryptografischen Hash-Algorithmus
identifiziert, der zur Berechnung der Hashes in diesem Merkle-Baum verwendet wird, entnommen aus der C2PA-Hash-
Algorithmus-Identifikatorliste. Wenn dieses Feld fehlt, wird der Hash-Algorithmus aus dem Wert „alg” der umgebenden
Struktur übernommen, wie er durch diese Struktur definiert ist. Wenn beide vorhanden sind, wird das Feld in dieser
Struktur verwendet. Wenn an keiner dieser Stellen ein Wert vorhanden ist, ist diese Struktur ungültig; es gibt
keinen Standardwert.
   ? „initHash”: bstr, ; Bei fragmentierten MP4-Assets, die auf mehrere Dateien aufgeteilt sind, muss dieses
Feld vorhanden sein und enthält den Hash der gesamten Initialisierungssegmentdatei für Chunks, die von diesem
Merkle-Baum gehasht wurden, mit Ausnahme der im Ausschlussarray aufgeführten Boxen. Bei fragmentierten MP4-
Assets, die als einzelne flache MP4-Datei gespeichert sind, muss dieses Feld vorhanden sein und ist der Hash
aller Bytes vor dem ersten „moof”-Box, mit Ausnahme der im Ausschlussarray aufgeführten Boxen. Bei nicht
fragmentierten MP4-Dateien darf dieses Feld nicht vorhanden sein.
   „hashes”: [1* bstr], ; Ein geordnetes Array, das eine einzelne Zeile des Merkle-Baums darstellt, bei der es
sich um die unterste Zeile, die Wurzelzeile oder eine beliebige Zwischenzeile handeln kann. Die Tiefe der
Zeile wird durch die Anzahl der Elemente in diesem Array impliziert (berechnet).
   ? „fixedBlockSize”: uint, ; Bei nicht fragmentierten MP4-Assets, bei denen die mdat-Box stückweise
validiert wird, kann dieses Feld vorhanden sein. Dieses Feld ist die nicht negative Größe in Byte eines
bestimmten Blattknotens im Merkle-Baum. Bei fragmentierten MP4-Dateien ist dieses Feld nicht vorhanden.
   ? „variableBlockSizes”: [1* int], ; Bei nicht fragmentierten MP4-Assets, bei denen die mdat-Box stückweise
validiert wird, kann dieses Feld vorhanden sein. Jeder Eintrag im Array entspricht der nicht negativen Größe
in Byte eines bestimmten Blattknotens im Merkle-Baum. Die Anzahl der Elemente entspricht `count` und die
Summe der Werte entspricht der Größe der Nutzlast von mdat. Bei fragmentierten MP4-Dateien ist dieses Feld
nicht vorhanden.
}
```

Ein Beispiel in CBOR-Diagnoseschrift (RFC 8949, Abschnitt 8) für eine monolithische MP4-Datei, bei der die mdat-Box als Einheit validiert wird, ist unten dargestellt:

```
{
   "hash": b64'EiAuxjtmax46cC2N3Y9aFmBO9Jfay8LEwJWzBUtZ0sUM8gA=', "name": "Beispiel
   `c2pa.hash.bmff.v2` Assertion",
   „exclusions”: [
      {
         „data”: [
            {
               „value”: b64'2P7D1hsOSDySl1goh37EgQ==', „offset”: 8
            }
```

```
        ],
        „xpath”: „/uuid”
      },
      {
        „xpath”: „/ftyp”
      },
      {
        „xpath”: „/mfra”
      },
      {
        „xpath”: „/moov[1]/pssh”
      },
      {
        „xpath”: „/emsg”,
        „data”: [
           {
             „Wert“: b64'r3avWCpXHkmKHATFsV0Q5g==', „Offset“: 20
           }
        ]
      }
   ]
}
```

Ein Beispiel in CBOR-Diagnoseschrift (RFC 8949, Abschnitt 8) für eine Ressource, die aus fragmentierten MP4-Dateien besteht, ist unten aufgeführt:

```
{
   „alg”: „sha256”,
   "name": "Beispiel `c2pa.hash.bmff.v3`-Assertion für fMP4", "merkle": [
      {
        „count”: 23,
        „Hashes”: [ b64'HvWZOxKMfkSatRAygs8DJfnEEcN/G1BNi359NdIDxbQ=',
b64'HvWZOxKMfkSatRAygs8DJfnEEcN/G1BNi359NdIDxbQ=' ],
        „localId”: 19,
        „initHash”: b64'Hf0IgeqbL0m+FTTLpUWwsDGR8pvhUR1AlwvaXjQ0qGY=', „uniqueId”:
        17
      },
      {
        „count”: 69,
        "hashes": [ b64'9Zk7Eox+RJq1EDKCzwMl+cQRw38bUE2Lfn010gPFtB0=',
b64'9Zk7Eox+RJq1EDKCzwMl+cQRw38bUE2Lfn010gPFtB0=', b64'mTsSjH5EmrUQMoLPAyX5xBHDfxtQTYt+fTXSA8W0Hf0=',
b64'mTsSjH5EmrUQMoLPAyX5xBHDfxtQTYt+fTXSA8W0Hf0=', b64'OxKMfkSatRAygs8DJfnEEcN/G1BNi359NdIDxbQd/Qg=' ],
        „localId”: 38,
        „initHash”: b64'Hf0IgeqbL0m+FTTLpUWwsDGR8pvhUR1AlwvaXjQ0qGY=', „uniqueId”:
        34
      },
      {
        „count”: 46,
        „Hashes“: [ b64'OxKMfkSatRAygs8DJfnEEcN/G1BNi359NdIDxbQd/Qg=' ], „localId“: 57,
        „initHash”: b64'Hf0IgeqbL0m+FTTLpUWwsDGR8pvhUR1AlwvaXjQ0qGY=', „uniqueId”:
        51
      }
   ],
   „exclusions”: [
      {
```

```
        „data“: [
          {
            „value“: b64'2P7D1hsOSDySl1goh37EgQ==', „offset“:
            8
          }
        ],
        „xpath“: „/uuid“
      },
      {
        „xpath“: „/ftyp“
      },
      {
        „xpath“: „/mfra“
      },
      {
        „xpath“: „/moov[1]/pssh“
      },
      {
        „data“: [
          {
            „value“: b64'9Q==', „offset“: 5
          },
          {
            „Wert“: b64'UAJXD79SlkG9rfnmcsqTUA==', „Offset“:
            20
          },
          {
            „Wert“: b64'OxKM', „Offset“: 70
          }
        ],
        „flags“: b64'ZDNx',
        „xpath“: „/emsg“, „length“:
        200,
        „Teilmenge“: [
          {
            „length“: 7,
            „offset“: 5
          },
          {
            „length“: 28,
            „Offset“: 20
          },
          {
            „length“: 63,
            „Offset“: 45
          },
          {
            „length“: 112,
            „Offset“: 80
          }
        ],
        „version“: 1
      }
    ]
}


{
  „alg“: „sha256“,
  „name“: „Beispiel `c2pa.hash.bmff.v3`-Assertion für nicht fragmentierte MP4“,
  „merkle“: [
```

```
    {
      „count”: 3,
      „Hashes”: [ b64'HvWZOxKMfkSatRAygs8DJfnEEcN/G1BNi359NdIDxbQ=',
b64'HvWZOxKMfkSatRAygs8DJfnEEcN/G1BNi359NdIDxbQ=' ], "variableBlockSizes": [ 100,
      30, 20 ],
      „localId”: 19,
      „initHash”: b64'Hf0IgeqbL0m+FTTLpUWwsDGR8pvhUR1AlwvaXjQ0qGY=', „uniqueId”: 17
    }
  ],
  „exclusions”: [
    {
      „data”: [
        {
          „value”: b64'2P7D1hsOSDySl1goh37EgQ==', „offset”:
          8
        }
      ],
      „xpath”: „/uuid”
    },
    {
      „xpath”: „/ftyp”
    },
    {
      „xpath”: „/mfra”
    },
    {
      „xpath”: „/moov[1]/pssh”
    },
    {
      „data”: [
        {
          „value”: b64'9Q==', „offset”: 5
        },
        {
          „Wert“: b64'UAJXD79SlkG9rfnmcsqTUA==', „Offset“:
          20
        },
        {
          „Wert“: b64'OxKM', „Offset“: 70
        }
      ],
      „flags”: b64'ZDNx',
      „xpath”: „/emsg”, „length”:
      200,
      „Teilmenge“: [
        {
          „length”: 7,
          „offset”: 5
        },
        {
          „length”: 28,
          „Offset“: 20
        },
        {
          „length“: 63,
          „offset”: 45
        },
        {
          „length”: 112,
          „Offset“: 80
```

```
          }
        ],
        „version”: 1
      }
    ]
}
```

Eine Pseudocode-Implementierung dieses Algorithmus finden Sie in Beispiel 6, „Pseudocode für BMFF-basierte Hash-Assertion”.

Beispiel 6. Pseudocode für BMFF-basierte Hash-Assertion

```
offset = 0
While (Offset < Länge der Datei)
     Beginnend bei Offset das erste Byte der ersten Box suchen, das mit einem Eintrag im
Ausschlussarray übereinstimmt, und dieses first_excluded_byte nennen
          Wenn keine solche Box gefunden wird, setze first_excluded_byte = Länge der Datei
     Bestimme die Länge dieser Box und nenne sie excluded_byte_count
          Wenn kein solcher Block gefunden wurde, setzen Sie excluded_byte_count = 0
     Fügen Sie zum Hash alle Bytes zwischen Offset und first_excluded_byte minus eins (einschließlich)
hinzu
     Wenn first_excluded_byte < Länge der Datei und es gibt ein Teilarray innerhalb der Ausschlussmenge, das den
Wert von first_excluded_byte bestimmt hat
          setze next_included_begin = first_excluded_byte
          Für jeden Eintrag im Teilarray innerhalb der Ausschlussmenge, die den Wert von
first_excluded_byte bestimmt hat
               Setze next_excluded_begin = Offset-Feld dieses Teilmengen-Array-Eintrags plus first_excluded_byte
               Wenn next_excluded_begin > next_included_begin
                    Füge zum Hash alle Bytes zwischen next_included_begin und
next_excluded_begin minus eins (einschließlich) hinzu
               Setze next_included_begin = Länge dieses Teilmengen-Array-Eintrags plus next_excluded_begin
          Wenn next_included_begin < first_excluded_byte + excluded_byte_count Füge dem
               Hash alle Bytes zwischen next_included_begin und
first_excluded_byte + excluded_byte_count minus eins (einschließlich) Setzen
     Sie offset = first_excluded_byte + excluded_byte_count
```

Ein Beispiel für die Generierung eines Hash für die Merkle-Map finden Sie in Beispiel 7, „Ein vorgeschlagenes Beispiel für eine Merkle-Map”.

Beispiel 7. Ein vorgeschlagenes Beispiel für eine Merkle-Map

```
Wenn die Felder „fixedBlockSize” und „variableBlockSizes” nicht vorhanden sind
     Fügen Sie zum Hash alle Bytes zwischen begin_address und der letzten Adresse der mdat-Nutzlast
hinzu. Wenn das Feld „fixedBlockSize” vorhanden ist, das Feld „variableBlockSizes” jedoch nicht
     Während (1)
          next_address = begin_address + fixedBlockSize
          Wenn next_address > letzte Adresse der mdat-Nutzlast next_address = letzte
               Adresse der mdat-Nutzlast plus eins hash_complete = true
          Fügen Sie zum Hash alle Bytes zwischen begin_address und next_address minus eins (einschließlich)
hinzu
          Wenn hash_complete wahr ist,
               brechen
          begin_address = next_address
```

```
Wenn das Feld „variableBlockSizes” vorhanden ist und das Feld „fixedBlockSize” nicht vorhanden ist
     Für (blockSize in variableBlockSizes) next_address =
          begin_address + blockSize
          Wenn next_address > letzte Adresse der mdat-Nutzlast next_address = letzte
               Adresse der mdat-Nutzlast plus eins hash_complete = true
          Füge zum Hash alle Bytes zwischen begin_address und next_address minus eins (einschließlich)
hinzu
          Wenn hash_complete wahr ist,
               brechen
          begin_address = next_address
```

18.6.4. Ausschlusslistenprofile

18.6.4.1. Allgemeines

In diesem Abschnitt wird eine Reihe vordefinierter, benannter Profile von Erweiterungslisten beschrieben.

18.6.4.2. Grundlegendes Profil

Typische zeitunabhängige Medien (z. B. Standbilder) und zeitabhängige Medien (z. B. Videos mit oder ohne Audiospuren, unabhängig davon, ob sie fragmentiert sind oder nicht) müssen nur die in den Anforderungen für Ausschlusslisten aufgeführten obligatorischen Ausschlüsse enthalten.

18.6.5. Fragmentiertes BMFF-Entitätsdiagramm

Abbildung 15, „Fragmentiertes BMFF-Entitätsdiagramm“, zeigt die Beziehung zwischen C2PA-Objekten, die ein fragmentiertes BMFF-Manifest bilden.

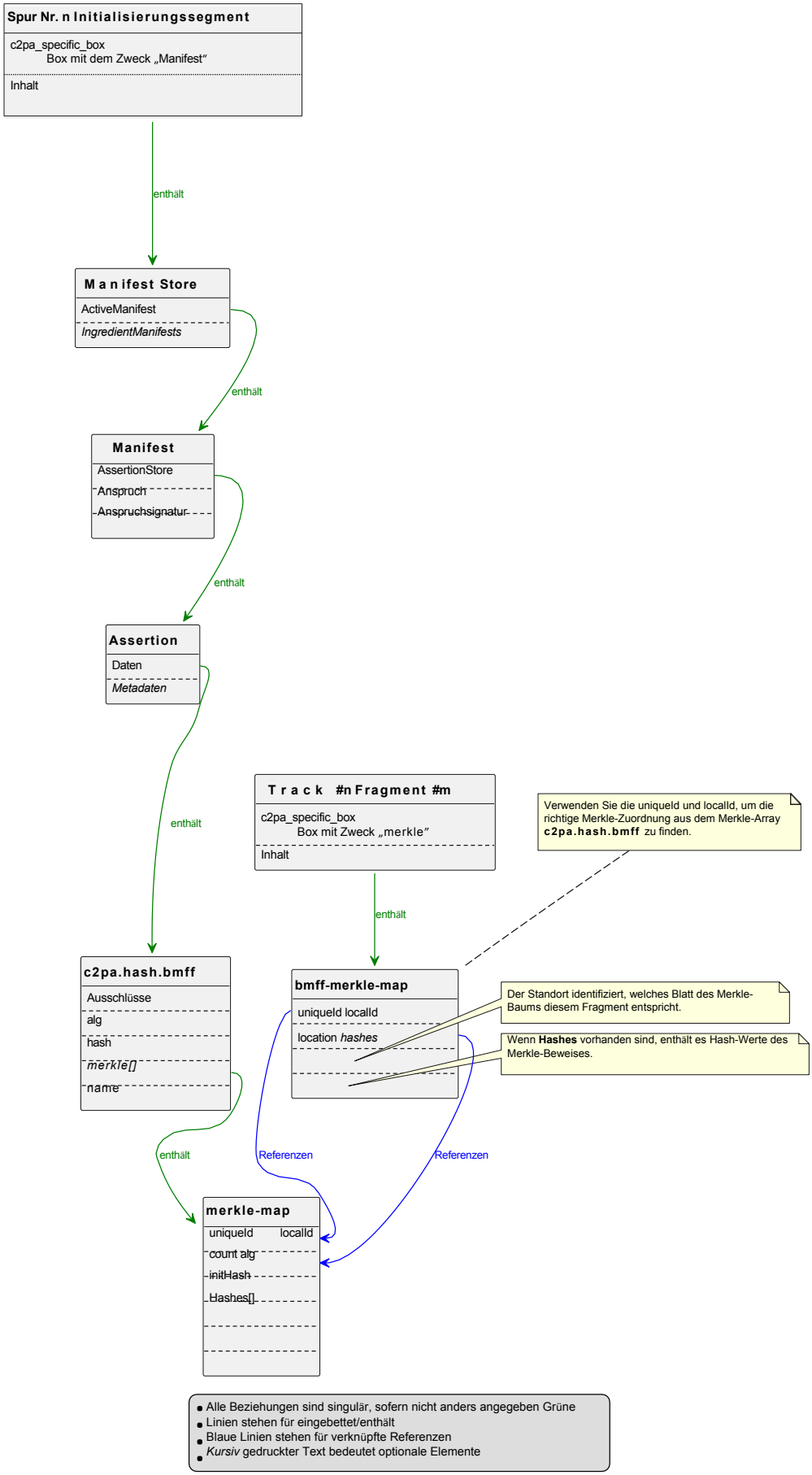

Abbildung 15. Fragmentiertes BMFF-Entitätsdiagramm

18.6.6. Validierung

Um einen bestimmten Chunk zu validieren, muss zunächst der `initHash` des Merkle-Map-Feldes über das entsprechende Initialisierungssegment validiert werden. Anschließend muss der richtige Eintrag im Hashes-Array des Merkle-Map-Feldes gefunden und anhand des Hashs der Chunk-Daten validiert werden. Bei Bedarf muss dieser Hash mithilfe des Merkle-Proofs aus den im `bmff-merkle-map` des Chunks angegebenen `Hashes` abgeleitet werden.

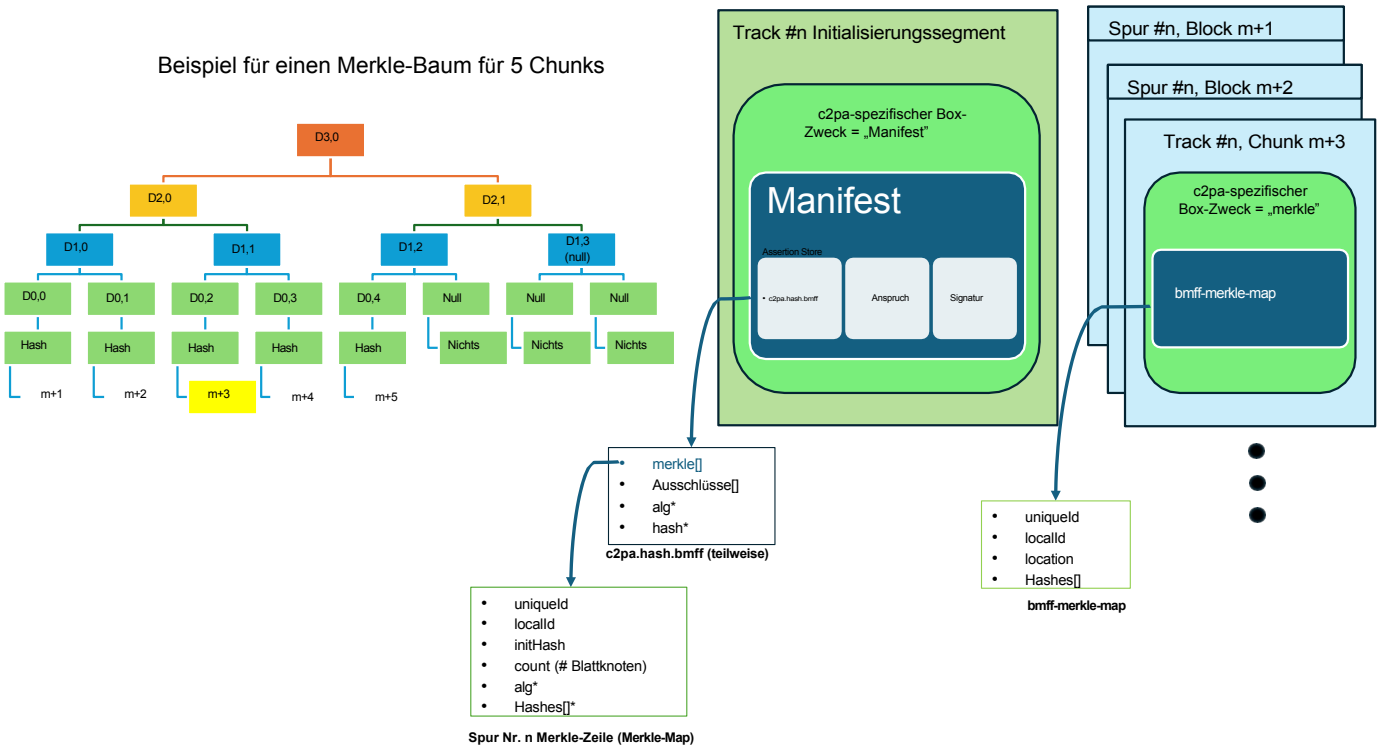

Abbildung 16. Validierung des Initialisierungssegments und Beispiel für die Daten eines Chunks

Um den Track-Chunk **m+3** zu überprüfen, müssen Sie zunächst das entsprechende Initialisierungssegment überprüfen. Das c2pa-spezifische Manifestfeld im Initialisierungssegment jedes Tracks enthält den Manifest-Speicher. Wenn das Asset mehrere Initialisierungssegmente enthält, muss der Manifest-Speicher in jedem Segment identisch sein. Auf diese Weise können Validatoren überprüfen, ob ein Track zu einem größeren Satz gehört. Die `c2pa.bmff.`hash-Assertion des aktiven Manifests enthält ein Merkle-Feld mit einem Array von Merkle-Map-Objekten, eines pro Track.

18.6.6.1. Schritte

1. Beziehen Sie aus der `bmff-merkle-map` im c2pa-spezifischen Merkle-Feld des Chunks die Werte `uniqueId` und `localId`. Verwenden Sie die `uniqueId` und `localId`, um eine passende `Merkle-Map` aus dem Merkle-Array der `c2pa.bmff.`hash-Assertion im Initialisierungssegment zu finden.
2. Wenn der Hash des Init-Segments unter Verwendung der `c2pa.bmff.hash-Ausschlüsse` und des `Merkle-Map-Algorithmus` mit dem
   `initHash` innerhalb der `Merkle-Map entspricht`, die Sie gerade gefunden haben, ist das Initialisierungssegment verifiziert.

HINWEIS	Die Parameter `alg` & `hash` auf der obersten Ebene der `bmff-hash-map` werden für monolithisches MP4 verwendet, während `alg` & `hashes` in der `Merkle-Map` für fragmentiertes MP4 verwendet werden.

Um die Überprüfung von Chunk **m+3** abzuschließen: Wir betrachten `die` in Schritt 1 gefundene `Merkle-Map` von Track #n, die in diesem Beispiel die Zeile 2 des Merkle-Baums enthält – **D2,0** und **D2,1**.

3. Hash-Chunk **m+3** unter Verwendung des `Ausschluss`-Arrays `c2pa.hash.bmff` und des `Algorithmus` aus der `Merkle-Map`, was zu **D0,2(abgeleitet)**.

4. Das `bmff-merkle-map`-Hash-Array (Merkle-Proof) von Chunk m+3 enthält den Hash von Chunk m+4 **(D0,3)** und den Hash-Wert **D1,0** aus Zeile eins.

5. Hash **D0,2(abgeleitet)** und **D0,3,** um **D1,1(abgeleitet)** zu erhalten. Hash **D1,0** mit **D1,1(abgeleitet),** um **D2,0(abgeleitet)** zu erhalten.

6. Wenn **D2,0(abgeleitet)** = **D2,0**, wie im Assertion-Merkle-Map-Hash-Parameter gespeichert, und das entsprechende Initialisierungssegment in Schritt 2 verifiziert wurde, dann wurde Chunk m+3 verifiziert.

18.7. Allgemeiner Box-Hash

18.7.1. Beschreibung

Ein Anspruchgenerator sollte eine allgemeine Box-Hash-Assertion verwenden, um die Integrität von Assets zu überprüfen, deren Formate ein nicht auf BMFF basierendes Box-Format wie JPEG, PNG oder GIF verwenden, und zwar mit einer festen Bindung (d. h. einem kryptografischen Hash).

Eine allgemeine Box-Hash-Assertion muss die Bezeichnung `„c2pa.hash.boxes"` tragen. Eine solche Assertion besteht aus einem Array von Strukturen, von denen jede eine oder mehrere Boxen (mit ihrem Namen/ihrer Kennung) und einen Hash enthält, der die Daten dieser Boxen (und alle möglichen Daten, die in der Datei zwischen ihnen vorhanden sein können) abdeckt, zusammen mit dem für das Hashing verwendeten Algorithmus. Die Boxen müssen in der Assertion in derselben Reihenfolge erscheinen, in der sie im Asset vorkommen, einschließlich der Box, die das C2PA-Manifest enthält. Wenn im Asset weitere Boxen vorhanden sind, die nicht ausdrücklich in dieser Assertion enthalten sind, oder wenn die Boxen in der falschen Reihenfolge erscheinen, wird das Manifest während der Validierung gemäß Abschnitt 15.12.3, „Validieren eines allgemeinen Box-Hashs", abgelehnt.

Ein Feld kann auch ein `ausgeschlossenes` Feld enthalten, bei dem es sich um einen booleschen Wert handelt, der angibt, ob ein Validierer dieses Feld (und den zugehörigen Hash) während der Validierung ignorieren kann. Wenn dieses Feld fehlt oder vorhanden ist und seinen Wert `„false"` hat, muss das Feld gehasht und die Werte verglichen werden. Bei Boxen, die ein `ausgeschlossenes` Feld mit dem Wert `„true"` haben, sollte der Anspruchgenerator einen genauen Hashwert angeben, um die Kompatibilität mit älteren Validatoren zu gewährleisten, die das `ausgeschlossene` Feld nicht erkennen. Wenn der Anspruchgenerator nicht auf Abwärtskompatibilität achten muss, sollte er die Binärzeichenfolge `00` (ein einzelnes Byte mit dem Wert 0) als Hashwert schreiben.

Wenn es mehrere Instanzen desselben Box-Typs gibt, wie z. B. mehrere APP1-Segmente in einer JPEG-1-Datei, muss jede Instanz separat in der Assertion aufgeführt werden. JPEG-Segmente, die Fragmente mit derselben Segmentkennung sind, werden ebenfalls als separate Boxen aufgeführt, mit Ausnahme der Segmente, die den C2PA Manifest Store bilden (wie unten beschrieben).

Die Erstellung der Hashwerte wird in Abschnitt 13.1, „Hashing", beschrieben, und der Wert muss im Hashfeld vorhanden sein. Der Hashwert für einen Bereich von Boxen wird vom Beginn der ersten Box (im Bereich) bis zum Ende der letzten Box (im Bereich) berechnet. Dies umfasst alle beliebigen Bytes, die zwischen den Boxen vorhanden sein können.

HINWEIS Bei Verwendung eines Bereichs von Feldern sind alle Daten zwischen dem Beginn des ersten Feldes und dem Ende des letzten Feldes

im Hash enthalten. Bei separater Auflistung der einzelnen Boxen werden jedoch keine zusätzlichen Daten berücksichtigt, sondern nur die Daten innerhalb der aufgelisteten Box.

Die Box, die den C2PA Manifest Store enthält (z. B. `caBX` für PNG oder `21FF` für GIF), muss ebenfalls in einem eigenen Array aufgeführt werden. Um sie eindeutig als C2PA Manifest-Box zu identifizieren, muss sie den Namen `C2PA` haben und der Wert des `Hash` muss die Binärzeichenfolge `00` (ein einzelnes Byte mit dem Wert 0) sein. Der C2PA Manifest Store muss als einzelner Kasten dargestellt werden, selbst im Fall einer JPEG-Datei, bei der der Kasten über mehrere APP11-Markersegmente fragmentiert ist.

HINWEIS

Da Validatoren häufig in Kombination mit der Ausgabe von Dateiparsern verwendet werden, ist es aus Sicherheitsgründen empfehlenswert,
Hash-Werte für den gesamten Dateiinhalt außerhalb des C2PA Manifest Store. Dadurch wird die Integrität der Medien und des verknüpften Manifests sichergestellt.

Der Pad-Wert muss immer vorhanden sein und muss eine mit Nullen aufgefüllte Byte-Zeichenkette sein, es sei denn, er wurde während der Mehrfachdurchlaufverarbeitung durch etwas anderes ersetzt. In diesem Fall darf kein `Pad` vorhanden sein.

HINWEIS

In Abschnitt 10.4, „Mehrstufige Verarbeitung“, wird beschrieben, wie die richtigen Werte eingegeben und die Auffüllung angepasst werden.

Eine allgemeine Box-Hash-Assertion darf nicht in einer Cloud-Daten-Assertion erscheinen.

18.7.2. Spezielle Behandlung von mehrteiligen Assets

Zur Unterstützung von Dateiformaten, die aus mehreren Teilen bestehen (wie in Abschnitt 18.9, „Multi-Asset-Hash“, beschrieben), wird eine zusätzliche logische Box für Fälle definiert, in denen die Daten eines oder mehrerer Teile nach den boxbasierten Daten des primären Teils kommen. Diese Box muss `mit c2pa.after` (für beliebige Daten jenseits des Endes der Box-Struktur) gekennzeichnet sein. Die Box `c2pa.after` ist, sofern vorhanden, die letzte aufgeführte Box, und ihr Hash wird aus dem Byte berechnet, das auf die letzte Box folgt, bis zum Ende der physischen Datei.

Die Hard-Binding-Assertion, die das gesamte Asset abdeckt, ist die einzige Assertion, die eine `c2pa.after-Box` enthalten kann.
Box enthalten kann. Eine Hash-Assertion für einen einzelnen Teil darf nur den Inhalt dieses Teils selbst abdecken, nicht jedoch andere Teile.

18.7.3. Behandlung bestimmter Formate

18.7.3.1. JPEG-spezifische Handhabung

Bei der Arbeit mit JPEG wird die APP11-Box für andere Standards als C2PA (d. h. JPEG 360) verwendet. In diesen Fällen müssen alle Nicht-C2PA-APP11-Boxen in die Liste der gehashten Boxen aufgenommen werden. Die APP11-Boxen, die den C2PA-Manifest-Speicher enthalten, müssen mit `C2PA` gekennzeichnet werden. Alle anderen Boxen müssen mit dem Symbol aus ISO 10918-1:1994, Tabelle B.1, gekennzeichnet werden.

Der C2PA-Manifest-Speicher kann dadurch identifiziert werden, dass es sich um eine JUMBF-Superbox mit der Bezeichnung `c2pa` und einer JUMBF-Typ-UUID von `63327061-0011-0010-8000-00AA00389B71` handelt, wie in Abschnitt 11.1.4.2, „Manifest-Speicher“, beschrieben.

HINWEIS

Die Felder „Start of Scan“ und „Restart“ mit den Bezeichnungen `SOS` und `RST[n]` enthalten die entropiecodierten Segmente, die auf die jeweilige Markierung folgen.

Die MPF-Erweiterung (Multi-Picture Format) für JPEG kann ebenfalls mit dieser Methode unterstützt werden, indem alle in der Datei enthaltenen Boxen in der Reihenfolge ihres Auftretens aufgelistet werden, vorausgesetzt, dass zwischen dem `EOI` eines einzelnen Bildes und dem `SOI` des nächsten Bildes keine Daten vorhanden sind. Die Boxenliste würde die Segmente aus jedem einzelnen Bild im MPF der Reihe nach auflisten (`SOI`,
…, `EOI`, `SOI`, …, `EOI`, …). Wenn der Anspruchsgenerator jedoch plant, die MPF-Datei als mehrteiliges Asset zu behandeln, muss das Feld `„c2pa.after"` verwendet werden, um die zusätzlichen Teile zu hashen, die auf das `EOI` des ersten Einzelbildes (dem primären Teil) folgen.

18.7.3.2. PNG-spezifische Behandlung

Eine PNG-Datei beginnt immer mit einem 8-Byte-Header (`89 50 4E 47 0D 0A 1A 0A`). Um ihn einzufügen, verwenden Sie den speziellen Wert
`PNGh` als erstes Feld in der Liste der Felder und beginne mit dem Hashing ab dem ersten Byte des Bildes.

18.7.3.3. TIFF-spezifische Behandlung

Eine TIFF-Datei beginnt immer mit einem 8-Byte-Header. Um diesen einzufügen, verwenden Sie den Sonderwert `TIFh` als ersten Box in der Liste der Boxen.

Eine TIFF-Datei besteht aus einem oder mehreren IFDs (Bilddateiverzeichnissen), die „Super-Boxen" entsprechen. Jedes IFD enthält eine Reihe von Einträgen, die entweder als „IFD-Einträge" oder „TIFF-Felder" bezeichnet werden und die „Boxen" darstellen. Der `Box-Name` für jeden IFD-Eintrag ist der Wert des Tag-Feldes, der in eine Zeichenfolge seines Dezimalwertes umgewandelt wird.

Im Gegensatz zu anderen boxartigen Formaten sind die Daten eines IFD-Eintrags möglicherweise nicht im Eintrag enthalten (es sei denn, sie sind 4 Byte lang oder kleiner), sondern befinden sich an anderer Stelle in der Datei.

HINWEIS Die Länge der Daten eines IFD-Eintrags wird durch Multiplikation der Anzahl der Datenwerte (wie im Feld `„Count"` im IFD-Eintrag festgelegt) mit der Größe jedes Datenwerts (wie durch die Typ-Feld im IFD-Eintrag).

Der Hashwert eines IFD-Eintrags wird über die 12 Bytes des IFD-Eintrags berechnet. Wenn die Länge des IFD-Eintrags mehr als 4 Bytes beträgt, wird der Hashwert aus der Verkettung dieser 12 Bytes mit den Bytes der Datei berechnet, auf die der Eintrag verweist, beginnend mit dem im Feld `„Value Offset"` des IFD-Eintrags angegebenen Byte-Offset und über die gesamte Länge der Daten.

Bei einigen bekannten IFD-Einträgen – `StripOffsets` (273), `TileOffsets` (324) und `FreeOffsets` (288) – sind die vom IFD-Eintrag referenzierten Daten selbst eine Liste von Offsets zu den tatsächlichen Daten. In diesen Fällen sind die Daten, über die der Hash berechnet wird, die Verkettung der folgenden Elemente in der angegebenen Reihenfolge:

1. Die 12 Bytes des IFD,
2. Die Bytes, die bei `Value Offset` beginnen, mit einer Länge `von Count` mal der Größe von `Type,` die die Offsets enthält, und
3. Für jeden Offset in der Reihenfolge seines Auftretens werden die Bytes an diesem Offset mit der Länge angegeben, die durch den zugehörigen Byte-Zählungseintrag des Typs angegeben wird: `StripByteCounts` (279), `TileByteCounts` (325) und `FreeByteCounts` (289).

HINWEIS Die Bilddaten in einer TIFF-Datei würden daher durch diese Kombination aus „Offsets" und „Byte-Zählungen" gehasht werden.

TIFF unterstützt auch SubIFDs, einen IFD-Typ, der auf einen oder mehrere IFDs verweist und diese somit durch Verweis einbindet. Dazu gehören nicht nur der Typ `SubIFD` (330), sondern auch `EXIF` (34665), `GPS` (34853) und `Interoperability` (40965). Für alle diese IFD-Typen und alle anderen IFD-Typen, die auf diese Weise auf andere IFDs verweisen, sind die Daten, über die der Hash berechnet wird, die Verkettung der folgenden Elemente in der angegebenen Reihenfolge:

1. Die 12 Bytes des IFD,
2. Entweder:
    a. Wenn `N = 1`, die Bytes, die bei `Wert-Offset` beginnen und die Länge des `Typs` haben, der den Offset des referenzierten IFD enthält, oder
    b. Wenn `N > 1`, werden die Bytes, die bei `Wert-Offset` beginnen und die Länge des `Typs` haben, der den Offset zum Array der IFD-Offsets enthält, mit den Bytes verkettet, die bei diesem Offset beginnen und die Länge `Count` mal die Größe des `Typs` haben, der die Offsets zu jedem „verzweigten" IFD enthält.
3. Für jedes referenzierte IFD werden die Daten für den Hash für dieses IFD an diesem Offset wie in diesem Abschnitt angegeben rekursiv berechnet.

18.7.3.4. GIF-spezifische Behandlung

Der Hash einer Box, die ein „Packed Fields"-Attribut enthält, hasht auch die optionalen Daten, die durch dieses Attribut angegeben werden. Beispielsweise enthält der Bilddeskriptor den Block „Lokale Farbtabelle" und der logische Bildschirmdeskriptor den Block „Globale Farbtabelle", sofern diese vorhanden sind.

Für alle Boxen, die eine Blockbezeichnung enthalten, gilt folgende Namenskonvention: „<Blockbezeichnung>".

Für alle Erweiterungsblöcke gilt folgende Namenskonvention: „<Extension Introducer><Extension Label>". Die einzigen anderen

Blöcke, die nicht der oben genannten Namenskonvention entsprechen, sind:

- Die Kopfzeile wird mit „GIF89a" gekennzeichnet.
- Die tabellenbasierten Bilddaten werden mit „TBID" gekennzeichnet.
- Der logische Bildschirmdeskriptor wird mit „LSD" gekennzeichnet.

Beispiel:

- Header: „GIF89a".
- Trailer: „3B".
- Bilddeskriptor: „2C".
- Kommentarerweiterung: „21FE".

18.7.3.5. RIFF-spezifische Handhabung

RIFF-Dateiblöcke können in einer Baumstruktur beliebiger Tiefe verschachtelt sein. Die Wurzel dieser Struktur besteht aus einem oder mehreren L0-Blöcken, die jeweils die Blockkennung `RIFF` haben. Diese RIFF-Blöcke sind mit der folgenden Struktur definiert:

- Bytes 0-3: Blockkennung, immer `RIFF`.
- Bytes 4–7: Chunk-Länge (minus 8 Bytes für die Felder „Chunk-Kennung” und „Chunk-Länge”).
- Bytes 8-11: Medientyp-Kennung.
- Bytes 12-n: Chunk-Daten (alle **L1**-Chunks).

Nach der Medienartkennung kann der RIFF-Chunk einen oder mehrere L1-Sub-Chunks enthalten, die jeweils folgende Struktur aufweisen:

- Bytes 0-3: Chunk-Kennung.
- Bytes 4-7: Chunk-Länge (minus 8 Bytes für die Felder „Chunk-Kennung” und „Chunk-Länge”).
- Bytes 8-n: Chunk-Daten.
- Byte n+1: Auffüllbyte (falls erforderlich).

Eine spezielle Chunk-Kennung namens `LIST` kann verwendet werden, um Chunks innerhalb eines **L1**-Chunks zu verschachteln. Diese `LIST`-Chunks ahmen die Struktur von **L0**-RIFF-Chunks nach:

- Bytes 0-3: Chunk-Kennung, immer `LIST`.
- Bytes 4–7: Chunk-Länge (abzüglich 8 Bytes für die Felder „Chunk-Kennung” und „Chunk-Länge”).
- Bytes 8-11: Kennung des Listentyps.
- Bytes 12-n: Chunk-Daten (alle **L2**-Chunks).
- Byte n+1: Auffüllbyte (falls erforderlich).

Zur Berechnung eines allgemeinen Box-Hashwerts wird jeder L**0**-Chunk als einzelne Box mit einer Größe von genau 12 Byte und einem Boxnamen behandelt, der dem Medientyp-Identifikator (Bytes 8–11) entspricht. Jeder **Nicht-LIST**-L1-Chunk wird als Box mit einem Namen behandelt, der der Chunk-Kennung (Bytes 0–3) entspricht, und einem Inhalt, der sich vom Anfang der Chunk-Kennung (Byte 0) bis zum Auffüllbyte (falls vorhanden) erstreckt. Jeder `LIST` **L1**-Chunk wird als Box mit einem Namen behandelt, der dem Listentyp-Identifikator (Bytes 8-11) entspricht, und dessen Inhalt sich vom Anfang des Chunk-Identifikators (Byte 0) bis zum Füllbyte (falls vorhanden) erstreckt. Alle in einem `LIST` **L**1-Chunk (**L2** und höher) verschachtelten Chunks werden als Teil des Inhalts des `LIST` **L1**-Chunks behandelt und als einzelne Box gehasht.

In allen Fällen sind Füllbytes als Teil des Inhalts des vorhergehenden Chunks zu behandeln und in den Hash für dieses Feld aufzunehmen.

18.7.3.6. Fontspezifische Behandlung

Die Tabellen einer Schriftart entsprechen direkt den Hash-Boxen, einschließlich der C2PA-Tabelle. Tabellen

werden immer in der Reihenfolge aufgezählt, in der sie im Tabellenverzeichnis der Schriftart erscheinen.

Beachten Sie, dass das Tabellenverzeichnis selbst nicht Teil des Hash-Inhalts ist und daher von keiner Box abgedeckt wird.

Der Wert `„checkSumAdjustment”` ist bei der Berechnung des Hashwerts für die Box, die die

`Kopftabelle` enthält.

Die Gruppierung oder Nichtgruppierung von Font-Tabellen in der allgemeinen Box-Hash-Assertion liegt im Ermessen des Claim-Generators.

Hinweis: Für eine breite Verteilung erstellte Schriftarten können davon profitieren, wenn jede Tabelle einer einzelnen Box zugewiesen wird. Auf diese Weise wird der Hash auch dann weiterhin korrekt validiert, wenn die Schriftart in einem anderen Format neu gepackt wird. Im Gegensatz dazu können Systeme, die automatisch eine große Anzahl von Schriftarten generieren, wie z. B. ein Subsetter, Tabellen in weniger Boxen zusammenfassen, um die Verarbeitung zu optimieren. In diesem Fall werden die Box-Hashes nach einer Formatumwandlung möglicherweise nicht validiert, da zwischen den Tabellen Füllzeichen eingefügt werden.

Da Font-Verbraucher nicht auf Tabellen reagieren dürfen, die sie nicht erkennen, wird die bestehende Font-Verarbeitungsinfrastruktur erwarten, dass der Wert `„checkSumAdjustment”` der Kopfzeile den endgültigen Inhalt der `C2`PA-Tabelle selbst enthält, einschließlich aller lokalen Manifestdateien in ihrer Gesamtheit.

18.7.4. Schema und Beispiel

Das Schema für diesen Typ wird durch die Box-Map-Regel in der CDDL-Definition in CDDL für Box Hash definiert:

CDDL für Box Hash

```
box-map = {
   „boxes”: [1* box-hash-map],
   ? „alg”:tstr .size (1..max-tstr-length), ; Eine Zeichenfolge, die den kryptografischen Hash-Algorithmus
identifiziert, der zur Berechnung des Hash-Werts in dieser Assertion verwendet wird, entnommen aus der C2PA-
Hash-Algorithmus-Identifikatorliste. Wenn dieses Feld fehlt, wird der Hash-Algorithmus aus dem Wert „alg” der
umgebenden Struktur übernommen. Wenn beide vorhanden sind, wird das Feld in dieser Struktur verwendet. Wenn an
keiner dieser Stellen ein Wert vorhanden ist, ist diese Struktur ungültig; es gibt keinen Standardwert.
}

box-hash-map = {
   "names": [1* box-name], ; Ein Array von Zeichenfolgen, die die Box-Identifikatoren in der Reihenfolge ihres
Auftretens darstellen (z. B. `APP0`, `IHDR`)
   ? „alg”: tstr .size (1..max-tstr-length), ; Eine Zeichenfolge, die den kryptografischen Hash-Algorithmus
identifiziert, der zur Berechnung des Hash in dieser Assertion verwendet wird, entnommen aus der C2PA-Hash-
Algorithmus-Identifikatorliste. Wenn dieses Feld fehlt, wird der Hash-Algorithmus aus dem Wert „alg” der
umgebenden Struktur übernommen. Wenn beide vorhanden sind, wird das Feld in dieser Struktur verwendet. Wenn an
keiner dieser Stellen ein Wert vorhanden ist, ist diese Struktur ungültig; es gibt keinen Standardwert.
   „hash“: bstr, ; Byte-Zeichenkette des Hash-Werts
   ? „excluded”: bool, ; Ein boolescher Wert, der angibt, ob ein Validierer dieses Feld (und den zugehörigen
Hash) während der Validierung ignorieren kann. Wenn dieses Feld fehlt, wird das Feld gehasht und die Werte
werden verglichen.
   „pad“: bstr, ; Mit Nullen aufgefüllte Byte-Zeichenkette, die zum Auffüllen von Speicherplatz verwendet wird
}

box-name /= tstr .size (1..10)
```

Fünf Beispiele in CBOR-Diagnoseschrift (RFC 8949, Abschnitt 8) sind in Beispiel Box Hash aufgeführt:

1. JPEG;
2. PNG;
3. GIF;

4. DNG (TIFF) mit einem SubIFD;
5. TTF.

Beispiel-Box-Hash

```
// JPEG-Beispiel //
{
     „alg": „sha256", „boxes": [
          {
               „names": [„SOI", „APP0", „APP2"],
               „hash": b64'...', „pad":
               b64'',
          },
          {
               „names": [„C2PA"],
               „hash": b64'AA==',
               „pad": b64'',
          },
          {
               „names": [„DQT", „SOF0", „DHT", „SOS", „RST0", „RST1", „EOI"],
               „hash": b64'...', „pad":
               b64'',
          }
     ]
}

// PNG-Beispiel //
// ohne XMP-Box //
{
     „alg": „sha256", „boxes": [
          {
               "names" : ["PNGh", "IHDR"],
               "hash" : b64'...',
               "pad" : b64'',
          },
          {
               „names": [„C2PA"],
               „hash": b64'AA==',
               „pad": b64'',
          },
          {
               „names": [„sBIT"],
               „hash": b64'...', „pad":
               b64'',
          },
          {
               „names": [„iTXt"],
               „hash": b64'...',
               „excluded": true, „pad":
               b64'',
          },
          {
               „names": [„IDAT", „IEND"],
               „hash": b64'...', „pad":
               b64'',
          }
     ]
}
```

```
// GIF-Beispiel //
{
     „alg": „sha256", „boxes": [
          {
               „names": [„GIF89a", „LSD"]
               „hash": b64'...', „pad":
               b64'',
          },
          {
               „names": [„2C", „TBID", „2C", „TBID"],
               „hash": b64'...', „pad":
               b64'',
          },
          {
               „names": [„21FE"],
               „hash": b64'...', „pad":
               b64'',
          },
          {
               „names": [„21F9"],
               „hash": b64'...', „pad":
               b64'',
          },
          {
               „names": [„3B"],
               „hash": b64'...', „pad":
               b64'',
          },
     ]
}

// TIFF/DNG-Beispiel //
{
     „alg": „sha256", „boxes": [
          {
               „names": [„TIFh", „254", „256", „257", „258", „259", „262"],
               „hash": b64'...', „pad":
               b64'',
          },
          {
               „names": [„273", „277", „278", „279", „284"],
               „hash": b64'...', „pad":
               b64'',
          },
          {
               // Dies ist ein SubIFD, das ein sekundäres Bild enthält //
               „names": [„330", „254", „256", „257", „258", „259", „262", „277", „278", „279",
„284"],
               „hash": b64'...', „pad":
               b64'',
          },
          {
               „names": [„700", „34665"],
               „hash": b64'...', „pad":
               b64'',
          },
          {
               „names": [„C2PA"],
               „hash": b64'AA==',
               „pad": b64'',
```

```
          }
     ]
}

// TTF-Beispiel //
{
     „alg”: „sha256”, „boxes”: [
          {
               "names" : ["C2PA"],
               „hash“: b64'AA==',
               „pad”: b64'',
          },
          {
               „names“: [„PCLT“],
               „hash“: b64'...', „pad“:
               b64'',
          },
          {
               „names”: [„cmap”],
               „hash“: b64'...', „pad“:
               b64'',
          },
          {
               „names”: [„cvt”],
               „hash“: b64'...', „pad“:
               b64'',
          },
          {
               „names”: [„fpgm”],
               „hash“: b64'...', „pad“:
               b64'',
          },
          {
               „names”: [„gasp”],
               „hash“: b64'...', „pad“:
               b64'',
          },
          {
               „names”: [„glyf”],
               „hash“: b64'...', „pad“:
               b64'',
          },
          {
               „names”: [„head”],
               „hash“: b64'...', „pad“:
               b64'',
          },
          {
               „names”: [„hhea”],
               „hash“: b64'...', „pad“:
               b64'',
          },
          {
               „names”: [„hmtx”],
               „hash“: b64'...', „pad“:
               b64'',
          },
          {
               „names”: [„loca”],
               „hash“: b64'...', „pad“:
               b64'',
          },
```

```
          {
               "names" : ["maxp"],
               „hash“: b64'...', „pad“:
               b64'',
          },
          {
               „names”: [„name”],
               „hash“: b64'...', „pad“:
               b64'',
          },
          {
               „names”: [„post”],
               „hash“: b64'...', „pad“:
               b64'',
          },
          {
               „names”: [„prep”],
               „hash“: b64'...', „pad“:
               b64'',
          }
     ]
}
```

18.8. Sammlungsdaten-Hash

18.8.1. Beschreibung

In Arbeitsabläufen, bei denen im Voraus bekannt ist, dass sich das C2PA-Manifest auf eine Sammlung von Assets statt auf ein einzelnes Asset bezieht, wird die Sammlungsdaten-Hash-Assertion als Methode zur Angabe der festen Bindungen (d. h. kryptografischen Hashes) für die Assets in der Sammlung verwendet.

HINWEIS Es ist möglich, jeden Ordner des Trainingsdatensatzes eines KI-/ML-Modells zu beschreiben, indem jeder Ordner als separater Bestandteil des Manifests des vollständigen Trainingsdatensatzes behandelt wird.

Eine Hash-Assertion für Sammlungsdaten muss die Bezeichnung `c2pa.hash.collection.data` haben.

Eine Hash-Assertion für Sammlungsdaten darf nicht in einer Cloud-Daten-Assertion erscheinen.

18.8.2. Schema und Beispiel

Das Schema für diesen Typ wird durch die Regel `„collection-data-hash-map”` in der folgenden CDDL-Definition definiert:

```
; Ein Array von URIs und den zugehörigen Hashes
$collection-data-hash-map /= { „uris”: [1*
   uri-hashed-data-map],
   „alg”: tstr .size (1..max-tstr-length), ; Eine Zeichenfolge, die den kryptografischen Hash-Algorithmus
identifiziert, der zur Berechnung des Hash-Werts für jeden Eintrag des Arrays „uris” verwendet wird und aus der
C2PA-Hash-Algorithmus-Identifikatorliste stammt.
     ? „zip_central_directory_hash” : bstr,
}

; Die Datenstruktur, die zum Speichern einer Referenz auf eine URI und deren Hash verwendet wird.
$uri-hashed-data-map /= {
```

```
  „uri”: relative-url-type, ; Relative-URI-Referenz „hash”: bstr, ;
  Byte-Zeichenkette, die den Hashwert enthält
  ? „size”: size-type, ; Anzahl der Datenbytes
  ? „dc:format”: format-string, ; IANA-Medientyp der Daten
  ? „data_types”: [1* $asset-type-map], ; zusätzliche Informationen zum Datentyp
}

; mit CBOR Kopf (#) und Ende ($) werden in regulären Ausdrücken eingeführt, sodass sie nicht explizit benötigt
werden relative-url-type /= tstr .regexp "[-a-zA-Z0-9@:%._\\+~#=]{2,256}\\.[a-z]{2,6}\\b[-a-zA-Z0-
9@:%_\\+.~#?&//=]*"
```

Ein Beispiel in CBOR-Diagnoseschrift (RFC 8949, Abschnitt 8) ist unten aufgeführt:

```
// Beispiel für eine Liste von Remote-URLs //
{
     „alg”: „sha256”, „uris”: [
          {
               "uri": "photos/id/870.jpg"
               "hash": b64'+ddHMTUUEpuSF6dNaHFa9uFc1sSnY+O3l3MMPFvX5Ws=', "dc:format":
               "image/jpeg"
          },
          {
               „url”: „deepmind/bigbigan-resnet50/1”, „hash”:
               b64'...,
               „dc:format”: „application/octet-stream”, „data_types”: [
                    {
                         "type": "c2pa.types.generator",
                    },
                    {
                         „type”: „c2pa.types.model.tensorflow”, „version”:
                         „1.0.0”,
                    },
                    {
                         „type”: „c2pa.types.tensorflow.hubmodule”, „version”:
                         „1.0.0”,
                    }
               ]
          }
     ]
}

// Beispiel für eine Liste von (relativen) Datei-URIs //
{
     „alg”: „sha256”, „uris”: [
          {
               "uri": "image1.png",
               "hash": b64'U9Gyz05tmpftkoEYP6XYNsMnUbnS/KcktAg2vv7n1n8='
          },
          {
               „uri”: „document.pdf”,
               „hash”: b64'G5hfJwYeWTlflxOhmfCO9xDAK52aKQ+YbKNhRZeq92c='
          },
     ]
}

// Beispiel für eine Liste relativer Pfade innerhalb einer EPUB-Datei (die eine ZIP-Datei ist) //
{
```

```
    „alg": „sha256", „uris":
    [
        {
            „uri": „mimetype"
            "hash": b64'+ZXhhbXBsZSBvZiBhIGxpc3Qgb2YgcmVsYXRpdmUgc8=', "dc:format":
            "text/text"
        },
        {
            „uri": „META-INF/container.xml"
            „hash": b64'+ddHMTUUEpuSF6dNaHFa9uFc1sSnY+O3l3MMPFvX5Ws=', „dc:format":
            „text/xml"
        },
        {
            „uri": „cover_page.svg",
            „hash": b64'U9Gyz05tmpftkoEYP6XYNsMnUbnS/KcktAg2vv7n1n8='
        },
        {
            „uri": „chapter1.html",
            „hash": b64'G5hfJwYeWTlflxOhmfCO9xDAK52aKQ+YbKNhRZeq92c='
        },
    ]
}
```

18.8.3. Felder

Das Feld „uris" besteht aus einem Array von „uri-hashed-data-map"-Werten, die eine Sammlung von Assets darstellen. Das Feld „alg" ist wie in Abschnitt 13.1, „Hashing", beschrieben. Durch seine Verwendung wird sichergestellt, dass alle Inhaltselemente in der Liste mit demselben Algorithmus gehasht werden.

Für jede URI-Hash-Datenzuordnung muss das URI-Feld vorhanden sein und eine gültige relative URI enthalten. Alle URIs sind als relativ zum Speicherort des Manifests zu betrachten, unabhängig davon, ob dieser lokal, in einem Container (z. B. ZIP) oder in der Cloud liegt. Da eine relative URI Navigationselemente (z. B. `../`) enthalten kann, ist es möglich, auf Inhaltselemente zu verweisen, die sich nicht im selben Ordner wie das Manifest befinden – was ein Sicherheitsproblem darstellen würde. Ein Anspruchgenerator muss die URIs vor der Verwendung validieren oder bereinigen und sicherstellen, dass weder . noch `..` als Teil der URI erscheinen.

Das Hash-Feld ist eine Byte-Zeichenkette, die den gültigen Hash-Wert für das Inhaltselement darstellt, wie durch das Feld „alg" bestimmt
. Der Hash muss sich auf alle Bytes (von 0 bis n) des Inhaltselements beziehen – ohne Ausnahmen. Die übrigen

Felder sind identisch mit denen einer Inhaltsstoffangabe.

18.8.4. Hash-Berechnung für die Elemente der Sammlung

Jede Datei in der Sammlung muss einzeln mit dem im Feld „alg" definierten Hash-Algorithmus gehasht werden. Der resultierende Hash-Wert muss im Feld „hash" der URI-Hash-Datenzuordnung gespeichert werden, die mit der URI der Datei verknüpft ist.

Es müssen nicht alle Dateien einer bestimmten Hierarchie in eine Hash-Sammlung aufgenommen werden.

HINWEIS	Dies ist zwar nützlich, wenn Dateien vorhanden sind, die nicht gehasht werden müssen, bietet aber auch einem Angreifer die Möglichkeit, Dateien hinzuzufügen, ohne die Bindung ungültig zu machen.

18.9. Multi-Asset-Hash

18.9.1. Beschreibung

Es gibt eine Reihe von Dateiformaten, die aus mehreren Teilen bestehen, wobei jeder Teil selbst ein gültiges Dateiformat ist, z. B. wenn mehrere einzelne Bilder zu einer einzigen Datei zusammengefasst werden. Einige Beispiele hierfür sind:

- CIPA Multi-Picture Format (MPF)
- Android Ultra HDR-Format (das MPF verwendet)
- ISO 21496 HDR (das MPF verwendet)
- Android Motion Photo-Format (das kein MPF verwendet, aber neben MPF in derselben Datei existieren kann)

In einigen Fällen kann es wünschenswert oder sogar erforderlich sein, die Integrität jedes einzelnen Teils der Datei zu überprüfen, anstatt nur die Datei als Ganzes. Dementsprechend reicht der derzeitige Satz von Hard-Binding-Assertions nicht aus, um die Integrität jedes Teils separat zu überprüfen. Darüber hinaus können die einzelnen Teile über eigene C2PA-Manifeste verfügen, die aufgezeichnet werden müssen. Die Multi-Asset-Hash-Assertion wird verwendet, um diese Funktionalität bereitzustellen.

Ein weiterer Sonderfall ist, wenn ein einzelner Teil optional ist – d. h. wenn er im Rahmen eines Workflows, an dem kein vertrauenswürdiger Unterzeichner beteiligt ist, entfernt werden kann/wird –, aber dennoch die Möglichkeit bestehen soll, die Integrität des restlichen Teils der Datei zu überprüfen.

18.9.2. Details

Eine Multi-Asset-Hash-Assertion muss die Bezeichnung `c2pa.hash.multi-asset` tragen. Obwohl sie Hashes enthält und die Handhabung der Hard Binding modifiziert, wird sie nicht als Hard Binding betrachtet.

Eine Multi-Asset-Hash-Assertion darf nicht in einer Cloud-Daten-Assertion erscheinen.

Eine Multi-Asset-Hash-Assertion sollte nicht mit einem komprimierten Manifest verwendet werden.

HINWEIS	Es ist nicht klar, ob zwischen den beiden eine technische Inkompatibilität besteht, daher wird empfohlen, ihre gemeinsame Verwendung zu vermeiden, bis weitere Untersuchungen abgeschlossen sind.

Jeder Teil, einschließlich des primären Teils, muss als Teil-Hash-Map-Objekt innerhalb des Teile-Arrays dargestellt werden. Das Feld `„location”` muss ein `Locator`-Objekt enthalten, das die Position des Teils innerhalb der Datei beschreibt. Das Locator-Objekt muss entweder ein bmffBox-Feld oder `byteOffset-` und length-Felder enthalten. Das byteOffset-Feld muss den Byte-Offset (vom physischen Anfang der Datei) des Teils innerhalb der Datei enthalten, und das `length-Feld` muss die Länge des Teils in Byte enthalten. Das bmffBox-Feld muss die BMFF-Box des Teils enthalten, wenn der Teil im primären Teil enthalten ist, jedoch als spezifische BMFF-Box (z. B. `mpvd`, wie es von Motion Photo verwendet wird). Bei einem Teil, das durch ein bmffBox-Feld beschrieben wird, gilt der Inhalt des Teils nur als Nutzlast dieser Box, ohne den Box-Header.

Die Teile innerhalb des Teile-Arrays müssen in der Reihenfolge aufgelistet werden, in der sie in der Datei erscheinen, und die Teile müssen zusammenhängend sein, sich nicht überlappen und jedes Byte der Datei abdecken.

HINWEIS	Das Erscheinen in der Datei ist definiert als ihre sequenzielle Reihenfolge, wie sie sich vom Byte 0 bis zum letzten Byte der Datei befinden würden.

Das Feld „hashAssertion" muss einen gehashten URI zur Hash-Assertion für den Teil enthalten. Die Hash-Assertion eines Teils muss eine standardmäßige Hard-Binding-Assertion sein (z. B. c2pa.hash.data), aber das Label muss die Zeichenfolge .part und einen beliebigen Mehrfachinstanz-Identifikator enthalten. Beispiel: c2pa.hash.data.part 2.

HINWEIS	Durch Hinzufügen dieser Label-Suffixe wird deutlich, dass Hard-Binding-Assertions für Teile nicht berücksichtigt werden. Standard-Hard-Binding-Assertions betrachtet werden und daher mehrere Instanzen davon innerhalb eines C2PA-Manifests existieren können.

Das optionale Feld muss ein boolescher Wert sein, der angibt, ob das Vorhandensein des Teils optional ist – der Standardwert ist „false", wenn nicht vorhanden.

Wenn ein Teil über ein eigenes C2PA-Manifest verfügt, das nicht in diesem Teil enthalten ist (z. B. einzelne Frames in einem Multi-Frame-Asset), wird empfohlen, dieses C2PA-Manifest im Manifest-Speicher des Assets zu speichern und eine componentOf-Komponente zu erstellen, um darauf zu verweisen.

18.9.3. Schema und Beispiel

Das Schema für diesen Typ wird durch die Regel „multi-asset-hash-map" in der folgenden CDDL-Definition definiert:

```
multi-asset-hash-map = {
  „parts": [* part-hash-map]; Ein Array aus einem oder mehreren Hashes für einzelne Teile der mehrteiligen
Datei
}

byte-range-locator = (
  „byteOffset": uint         ; Der Byte-Offset des Teils innerhalb der Datei
  "length": uint             ; Die Länge des Teils
)

; Dies ist eine spezielle CDDL-Zuordnung von Auswahlmöglichkeiten (d. h., nur eine der folgenden Optionen kann
vorhanden sein)
locator-map = {
  byte-range-locator //          ; Der Byte-Offset und die Länge des Teils innerhalb der Datei
  „bmffBox": tstr            ; Ein XPath zur BMFF-Box des Teils
}

part-hash-map = {
  „location": locator-map, ; Die Position des Teils innerhalb der Datei „hashAssertion": $hashed-uri-
  map, ; hashed_uri zur Hash-Assertion des Teils
  ? „optional": bool, ; Wenn der Teil optional ist und verworfen werden kann
}
```

Ein Beispiel in CBOR-Diagnoseschrift (RFC 8949, Abschnitt 8) ist unten aufgeführt:

```
// Multi-Asset-Hash-Assertion //
// Das Asset (mit einer Größe von 33.333 Bytes) besteht aus einem JPEG-Teil in Bytes [0,11111) und einem
weiteren
// Teil in Bytes [11111,33333).
{
  „Teile": [
```

```
    {
      „location”: {
        „byteOffset”: 0,
        „length“: 11111
      },
      „hashAssertion“: „self#jumbf=c2pa.assertions/c2pa.hash.boxes.part“
    },
    {
      „location“: { „byteOffset“:
        11111,
        „length”: 22222
      },
      „hashAssertion”: „self#jumbf=c2pa.assertions/c2pa.hash.data.part”
    }
  ]
}

// c2pa.hash.boxes.part – Box-Hash für den ersten Teil des Assets //
{
  „alg”: „sha256”,
  „boxes”: [
    {
      "names" : ["SOI", "APP0", "APP2"],
      „hash”: b64'...', „pad”:
      b64'',
    },
    {
      „names”: [„C2PA”],
      „hash“: b64'AA==',
      „pad”: b64'',
    },
    {
      „names”: [„DQT”, „SOF0”, „DHT”, „SOS”, „RST0”, „RST1”, „EOI”],
      „hash“: b64'...', „pad“:
      b64'',
    }
  ]
}

// c2pa.hash.data.part – Daten-Hash für den zweiten Teil des Assets //
{
  „alg“: „sha256“,
  „pad”: '0000',
  „hash“: b64'...',
}

// c2pa.hash.boxes – Gesamt-Asset-Hash, der das gesamte zweiteilige Asset abdeckt //
{
  „alg”: „sha256”,
  „boxes”: [
    {
      "names" : ["SOI", "APP0", "APP2"],
      „hash”: b64'...', „pad”:
      b64''
    },
    {
      „names”: [„C2PA”],
      „hash“: b64'AA==',
      „pad”: b64''
    },
    {
      „names”: [„DQT”, „SOF0”, „DHT”, „SOS”, „RST0”, „RST1”, „EOI”],
      „hash“: b64„...“,
```

```
        „pad": b64''
      },
      {
        „names": [„c2pa.after"],
        „hash": b64'...',
        „pad": b64''
      }
    ]
}
```

Ein solches Beispiel für eine Multi-Asset-Hash-Assertion könnte in einem Bild enthalten sein, wie in [_multi_asset_hdr_image] gezeigt.

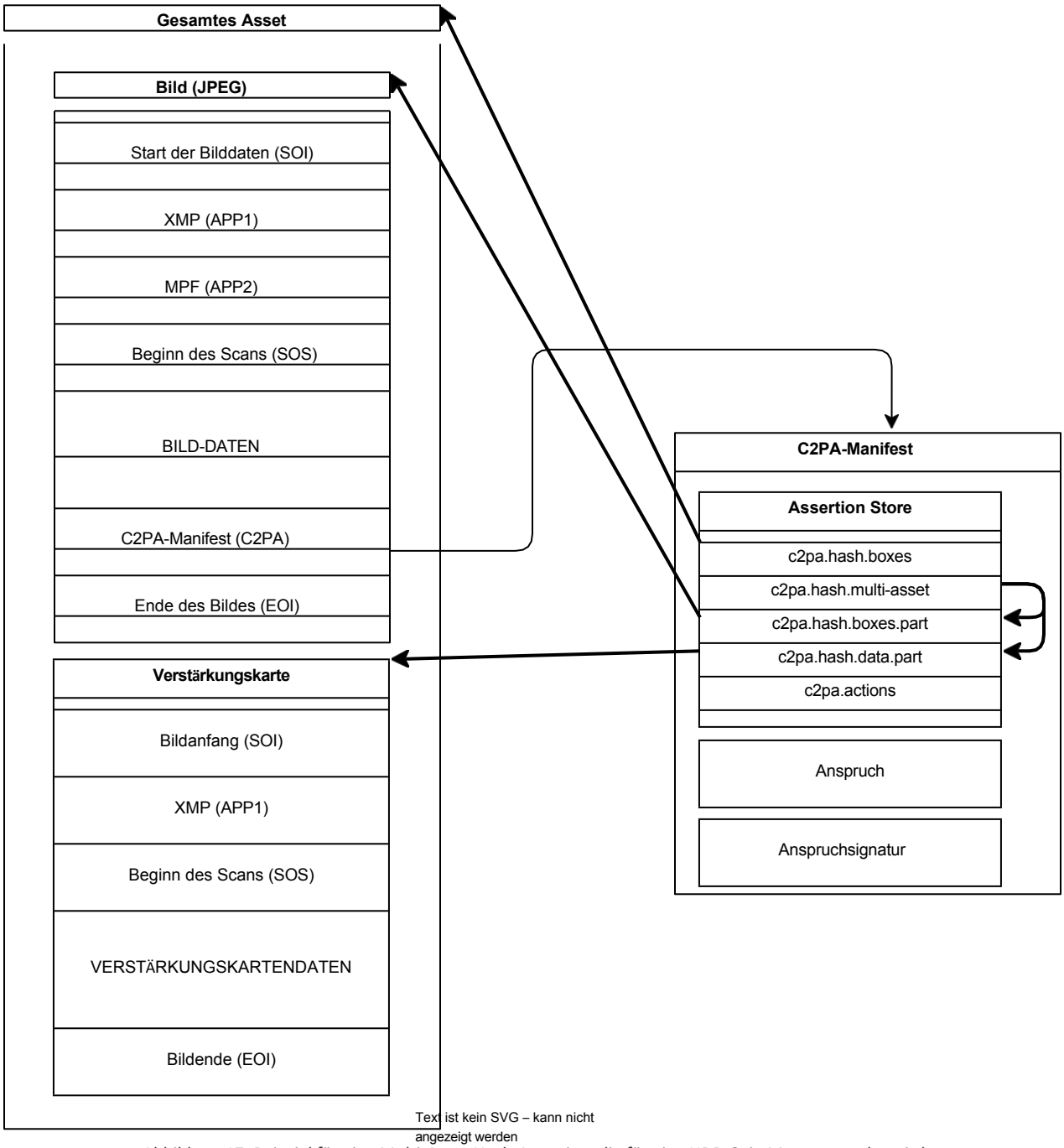

Abbildung 17. Beispiel für eine Multi-Asset-Hash-Assertion, die für eine HDR-Gain-Map verwendet wird

18.10. Weiche Bindung

18.10.1. Beschreibung

Wenn ein Anspruchgenerator eine weiche Bindung für den Inhalt des Vermögenswerts bereitstellt, muss dies mithilfe einer weichen Bindungsaussage beschrieben werden. Die Arten von weichen Bindungen, die in einer solchen Aussage erstellt und gespeichert werden können, sind in Abschnitt 18.10, „Weiche Bindung“, beschrieben.

Eine frühere Version dieser Spezifikation enthielt ein URL-Feld, um einen Verweis auf den Speicherort der Hash-Daten anzugeben, das jedoch nie verwendet wurde. Dieses Feld ist nun zugunsten der Asset-Referenz-Assertion veraltet. Claim-Generatoren dürfen dieses Feld nicht zu einer Soft-Binding-Assertion hinzufügen, und Verbraucher müssen das Feld ignorieren, wenn es vorhanden ist, außer dass dies keinen Einfluss auf die Einbeziehung des Feldes als Teil des zu validierenden Inhalts hat, wie in Abschnitt 15.10.3, „Assertion Validation” (Assertion-Validierung), beschrieben.

Eine frühere Version dieser Spezifikation enthielt ein Extent-Feld innerhalb des Scope-Feldes, um einen Teil des digitalen Inhalts, der von der Soft-Binding-Assertion abgedeckt wird, in einem algorithmusspezifischen Format zu beschreiben. Dieses Feld ist nun zugunsten des Region-Feldes veraltet. Anspruchgeneratoren dürfen dieses Feld nicht zu einer Soft-Binding-Assertion hinzufügen, und Verbraucher sollten das Feld ignorieren, wenn es vorhanden ist. Dies hat keinen Einfluss auf die Einbeziehung des Feldes als Teil des zu validierenden Inhalts, wie in Abschnitt 15.10.3, „Assertion Validation“ (Assertion-Validierung), beschrieben.

Eine Soft-Binding-Assertion muss die Bezeichnung `c2pa.soft-binding` haben.

18.10.2. Schema und Beispiel

Das Schema für diesen Typ wird durch die Regel „`soft-binding-map`“ in der folgenden CDDL-Definition definiert:

```
;Struktur der Objekte für Regionen von Interesse in Soft-Binding-Assertions an die für andere Zwecke
verwendete Struktur anpassen
;# Regionen von Interesse einbeziehen

;Die Datenstruktur, die zum Speichern einer oder mehrerer Soft-Bindings über einen Teil oder den gesamten
Inhalt des Assets verwendet wird
soft-binding-map = {
  „alg”: tstr, ; Eine Zeichenfolge, die den Soft-Binding-Algorithmus und die Version dieses Algorithmus
identifiziert, die zur Berechnung des Werts verwendet werden, entnommen aus der C2PA-Soft-Binding-
Algorithmusliste. Wenn dieses Feld fehlt, wird der Algorithmus aus dem Wert „alg_soft” der umgebenden Struktur
übernommen. Wenn beide vorhanden sind, wird das Feld in dieser Struktur verwendet. Wenn an keiner dieser
Stellen ein Wert vorhanden ist, ist diese Struktur ungültig; es gibt keinen Standardwert.
  „blocks”: [1* soft-binding-block-map],
  „pad”: bytes, ; mit Nullen aufgefüllte Byte-Zeichenkette, die zum Auffüllen von Speicherplatz verwendet wird
  ? „pad2”: Bytes, ; optionale mit Nullen aufgefüllte Byte-Zeichenkette zum Auffüllen von Speicherplatz
  ? „name”: tstr .size (1..max-tstr-length), ; (optional) eine für Menschen lesbare Beschreibung dessen, was
dieser Hash abdeckt
  ? „alg-params”: bstr, ; (optional) CBOR-Byte-Zeichenkette, die die Parameter des Soft-Binding-Algorithmus
beschreibt.
  ? „url”: uri, ; Nicht verwendet und veraltet.
}

soft-binding-block-map = { "scope":
  soft-binding-scope-map,
  „value”: bstr, ; CBOR-Byte-Zeichenkette, die in einem algorithmusspezifischen Format den Wert
```

```
Soft-Bindings, der über diesen Block digitaler Inhalte berechnet wurde.
}

soft-binding-scope-map = {
  ? „extent”: bstr, ;veraltet, CBOR-Byte-Zeichenkette, die in einem algorithmusspezifischen Format den Teil
des digitalen Inhalts beschreibt, über den der Soft-Binding-Wert berechnet wurde"
  ? „timespan”:soft-binding-timespan-map,
  ? „region”: region-map, ; CBOR-Objekt, definiert in regions-of-interest.cddl
}

soft-binding-timespan-map = {
  „start“: uint, ; Beginn des Zeitbereichs (in Millisekunden ab Medienstart), über den der Soft-Binding-Wert
berechnet wurde.
  „end”: uint,       ; Ende des Zeitbereichs (in Millisekunden ab Medienstart), über den der Soft-Binding-
Wert berechnet wurde.
}
```

Ein Beispiel in CBOR-Diagnoseschrift (RFC 8949, Abschnitt 8) ist unten aufgeführt:

```
{
  „alg“: „phash“,
  „pad”: h'00',
  „url“: 32(„http://example.c2pa.org/media.mp4“), „blocks“: [
    {
      „scope“: {
        „Zeitspanne“: {
          „Ende“: 133016
          „start”: 0,
        }
      },
      „value“: b64'dmFsdWUxCg=='
    },
    {
      „scope“: {
        „Zeitspanne“: {
          „Ende“: 245009
          „start“: 133017,
        }
      },

    }
  ]
}
```

18.10.3. Anforderungen

Der verwendete Soft-Binding-Algorithmus muss als Wert des Feldes „alg“ angegeben werden, und die Blöcke, auf die er angewendet wurde, müssen im Feld „blocks“ aufgeführt werden. Wenn der verwendete Algorithmus zusätzliche Parameter erfordert, müssen diese als Wert von „alg-params“ angegeben werden.

Das Feld „scope“ kann entweder ein Feld „region“ oder „timespan“ enthalten, um den Teil des digitalen Inhalts zu beschreiben, über den die weiche Bindung berechnet wurde. Das Feld „region“ enthält, sofern vorhanden, ein Objekt „region-map“ (wie in Abschnitt 18.2, „Regions of Interest“, definiert). Das Feld „timespan“ beschreibt, sofern vorhanden, das Zeitintervall, über das die weiche Bindung in Millisekunden ab Beginn des Inhalts berechnet wurde.

18.10.4. Liste der Soft-Binding-Algorithmen

Die Liste der Soft-Binding-Algorithmen ist eine maschinenlesbare Liste zulässiger Werte für das Feld „alg“. Das Feld „alg“ muss dem Feld „alg“ eines in dieser Liste enthaltenen Algorithmus entsprechen. Das Format der Felder „alg-params“ und „value“ ist algorithmusspezifisch und wird auf einer für Menschen lesbaren Informationsseite beschrieben, auf die über „informationalUrl“ innerhalb des Eintrags für „alg“ in der Liste verwiesen wird.

Die Liste wird als JSON-Dokument von der C2PA unter folgender Adresse gepflegt: https://github.com/c2pa-org/ softbinding-algorithm-list

Einträge in der Liste der Soft-Binding-Algorithmen, die ein veraltetes Feld mit dem Wert „true“ haben, gelten als veraltet und dürfen nicht zur Erstellung von Soft-Binding-Assertions in Manifesten verwendet werden. Als veraltet gekennzeichnete Soft-Binding-Algorithmen können zur Auflösung von Soft-Bindings verwendet werden, dies wird jedoch nicht empfohlen.

Das JSON-Schema für Einträge in der Liste der Soft-Binding-Algorithmen ist unten dargestellt:

```
{
     „$schema”: „https://json-schema.org/draft/2020-12/schema”, „type”:
     „object”,
     "properties": {
          "identifier": {
               "type": "integer",
               "minimum": 0,
               „maximum”: 65535,
               „Beschreibung“: „Diese Kennung wird zugewiesen, wenn der Soft-Binding-Algorithmus zur Liste
hinzugefügt wird.“
          },
          „veraltet“: { „Typ“:
               „Boolescher Wert“,
               „Standard“: false,
               „description“: „Gibt an, ob dieser Soft-Binding-Algorithmus veraltet ist.
Veraltete Algorithmen dürfen nicht zum Erstellen von Soft-Bindings verwendet werden. Veraltete Algorithmen
können zum Auflösen von Soft-Bindings verwendet werden, dieses Verhalten wird jedoch nicht empfohlen.“
          },
          „alg“: {
               „type“: „string“,

               "description": "Entitätsspezifischer Namespace, wie für C2PA-Assertions-Labels angegeben,
der mit dem Internet-Domänennamen für die Entität beginnen muss, ähnlich wie Java-Pakete definiert sind (z.
B. `com.example.algo1`, `net.example.algos.algo2`)"
          },
          „type”: {
               „type”: „string”, „enum”: [
                    „watermark“,
                    „fingerprint“
               ],
               „description”: „Art der durch diesen Algorithmus implementierten weichen Bindung.”
          },
          „decodedMediaTypes“: { „type“:
               „array“, „minItems“: 1,
               „Elemente“: {
                    „type“: „string“,
                    „enum“: [
                         „Anwendung“,
```

```
                         „audio“,
                         „image“,
                         „Modell“,
                         „Text“,
                         „Video“
                    ],
                    „Beschreibung“: „IANA-Top-Level-Medientyp (gerendert), für den dieser Soft-Binding-Algorithmus
gilt.“
               }
          },
          „encodedMediaTypes“: {
               „type“: „array“,
               „minItems“: 1,
               „items“: {
                    „Typ“: „Zeichenfolge“,
                    „description”: „IANA-Medientyp, für den dieser Soft-Binding-Algorithmus gilt, z. B.
application/pdf”,

\\+]+)*)$"
               }
          },
          „entryMetadata“: { „type“:
               „object“, „properties“:
               {
                    „description“: { „type“:
                         „string“,
                         „description“: „Für Menschen lesbare Beschreibung des Algorithmus.“
                    },
                    „dateEntered“: { „type“:
                         „string“,
                         „format“: „date-time“,
                         „description“: „Datum der Eingabe für diesen Algorithmus.“
                    },
                    „contact“: {
                         „type”: „string”,
                         „format“: „email“
                    },
                    „informationalUrl“: {
                         „type“: „string“,
                         „format“: „uri“,
                         „description“: „Eine Webseite mit weiteren Details zum Algorithmus.“
                    }
               },
               „required”: [
                    „description“,
                    „dateEntered“, „contact“,
                    „informationalUrl“
               ]
          },
          „softBindingResolutionApis”: { „type”:
               „array”,
               „Elemente“: {
                    „Typ“: „Zeichenfolge“,
                    „format”: „uri”
               },
               „description“: „Eine Liste der Soft Binding Resolution-APIs, die diesen
Algorithmus unterstützen.”
          }
     },
     „required”: [
          „identifier“, „alg“,
```

```
          „type“,
          „entryMetadata“
     ],
     „oneOf”: [
          {
               „required“: [
                    „decodedMediaTypes“
               ]
          },
          {
               „erforderlich“: [
                    „encodedMediaTypes”
               ]
          }
     ]
}
```

Ein JSON-Beispiel für einen Eintrag in der Liste der Soft-Binding-Algorithmen ist unten aufgeführt:

```
{
     „identifier”: 1,
     „deprecated”: false,
     „alg”: „com.example.product”, „type”:
     „watermark”, „decodedMediaTypes”: [
          „audio”,
          „video“,
          „text“,
          „image“
     ],
     „entryMetadata“: {
          „description“: „Foo Inc.s Wasserzeichen-Algorithmus Version 1.2“, „dateEntered“: „2024-04-
          23T18:25:43.511Z“,
          „contact“: „foo.bar@example.com “, „informationalUrl“:
          „https://example.com/wmdetails“
     },

     ]
}
```

Der eindeutige Name des Algorithmus wird im Feld „alg“ angegeben und entspricht der Zeichenfolge, die im Feld „alg“ einer Soft-Binding-Assertion verwendet werden soll, die diesen Algorithmus verwendet. Der Name muss den Anforderungen an den Namensraum entsprechen und den Eigentümer des Algorithmus repräsentieren. Jedem Algorithmus wird außerdem eine eindeutige numerische Kennung zugewiesen. Wenn verschiedene Versionen eines Algorithmus bereitgestellt werden, muss jede Version einen separaten Eintrag in der Soft-Binding-Algorithmusliste haben.

Der Typ des Algorithmus muss entweder „watermark“ (Wasserzeichen) oder „fingerprint“ (Fingerabdruck) sein, um anzugeben, dass es sich bei dem Algorithmus um ein unsichtbares Wasserzeichen oder einen Fingerabdruck handelt.

Der Verfallsstatus des Algorithmus wird im Feld „deprecated” angegeben. Ein Validator sollte keine Soft Bindings auflösen, die veraltete Algorithmen verwenden. C2PA-Manifeste dürfen nicht mit veralteten Soft Bindings geschrieben werden.

Der Eintrag in der Liste der Soft-Binding-Algorithmen muss eine Liste der unterstützten Medientypen entweder als „encodedMediaTypes“ oder als „decodedMediaTypes“ enthalten. Die unterstützten Medientypen für „decodedMediaTypes“ müssen einem oder mehreren der folgenden entsprechen

Die IANA-Medientypen der obersten Ebene umfassen: „application“, „audio“, „image“, „model“, „text“ und „video“. Die unterstützten Medientypen für `encodedMediaTypes` müssen einem oder mehreren der registrierten IANA-Subtypen eines `decodedMediaType` entsprechen, die im vorstehenden Satz aufgeführt sind. Diese IANA-Top-Level- und Subtypen sind unter https://www.iana.org/assignments/media-types/media-types.xhtml aufgeführt.

Zusätzliche Informationen müssen jedem Eintrag in der Liste der Soft-Binding-Algorithmen im Feld `entryMetadata` beigefügt werden. Dabei handelt es sich um eine für Menschen lesbare Beschreibung des Algorithmus (`description`) und das Datum, an dem er für die Aufnahme in die Liste der Soft-Binding-Algorithmen vorgeschlagen wurde (`dateEntered`).

Die Kontaktdaten des Eigentümers des Eintrags müssen als E-Mail-Adresse (`contact`, erforderlich) angegeben werden. Es muss eine Informations-URL (`informationalUrl`, erforderlich) angegeben werden, die auf eine für Menschen lesbare Seite verweist, auf der die Eigenschaften des Soft-Binding-Algorithmus beschrieben werden. Die Informationen auf dieser Seite unterliegen keinen Beschränkungen, können jedoch Details enthalten, z. B. wie das Wertefeld im Soft-Binding-Register zu interpretieren ist, das in einer algorithmusspezifischen Form codiert ist.

18.10.5. API zur Auflösung weicher Bindungen

Die Soft-Binding-Resolution-API ist eine Web-API, die eine Standardmethode zum Abrufen von C2PA-Manifest-Speichern von einem Soft-Binding-Resolution-API-Endpunkt bereitstellt, wenn ein Soft-Binding-Wert, eine Manifest-Kennung oder eine Ressource angegeben wird. Der Soft-Binding-Algorithmus-Listeneintrag kann eine Liste von URIs von Soft-Binding-Resolution-APIs im Feld `„softBindingResolutionApis“` enthalten. Wenn mehrere URIs angegeben sind, kann jede davon für eine Soft-Binding-Resolution verwendet werden.

Die API-Spezifikation und Dokumentation finden Sie hier.

18.10.5.1. Validieren von Soft-Binding-Übereinstimmungen

Eine häufige Verwendung für Soft Bindings ist das Auffinden des aktiven Manifests aus einem Manifest-Repository für ein Asset, dessen C2PA-Manifest fehlt oder ungültig ist.

Die Ermittlung des C2PA-Manifests muss unter Verwendung eines oder einer Kombination von Algorithmen erfolgen, die im Feld `„alg“` innerhalb der C2PA-Soft-Binding-Algorithmusliste angegeben sind. Die Liste wird von der C2PA als JSON-Dokument unter folgender Adresse gepflegt: https://github.com/c2pa-org/softbinding-algorithm-list

Wenn ein C2PA-Manifest in einem Manifest-Repository gefunden wird und dieses Manifest eine oder mehrere Soft-Binding-Assertions enthält, muss der Matcher sicherstellen, dass alle Soft-Binding-Assertions im gefundenen Manifest mit den Soft-Bindings übereinstimmen, die zur Durchführung der Erkennung verwendet wurden.

Eine Soft-Binding-Assertion gilt als Übereinstimmung, wenn sowohl die in der Assertion beschriebene Algorithmuskennung (`alg`) als auch der Wert (`value`) mit der Algorithmuskennung (`alg`) und dem Wert (`value`) übereinstimmen, die zur Durchführung des Abgleichs verwendet wurden. Der Abgleich erfolgt in der durch den angegebenen Algorithmus vorgeschriebenen Weise.

18.11. Cloud-Daten

18.11.1. Beschreibung

Es gibt Anwendungsfälle, in denen es besser ist, die Daten für die Assertion remote zu speichern, beispielsweise in der Cloud, als sie in das Asset einzubetten, insbesondere wenn es sich um große Datenmengen handelt. In solchen Fällen kann eine spezielle Art von Assertion verwendet werden, die als Verweis auf diese Informationen dient. Aus Gründen der Privatsphäre und Zuverlässigkeit sollten Daten, auf die über eine Cloud-Daten-Assertion verwiesen wird, als optional betrachtet werden: Ihr Inhalt sollte nicht im Rahmen der Manifestvalidierung abgerufen werden. Ein Validierer kann den Inhalt später abrufen, um einem anwendungsabhängigen Bedarf gerecht zu werden, beispielsweise zur weiteren Untersuchung der Herkunftsgeschichte.

Wenn Assertion-Metadaten als Teil einer anderen Assertion enthalten sind, wären auch diese Teil der Informationen, auf die eine Cloud-Daten-Assertion verweist. Es ist auch möglich, einzelne Assertion-Metadaten-Assertionen remote zu speichern, genau wie bei anderen Assertion-Typen.

Eine Cloud-Daten-Assertion muss die Bezeichnung `c2pa.cloud-data` tragen.

Eine Cloud-Daten-Assertion darf nicht auf eine Assertion mit der Kennzeichnung „`c2pa.hash.data`“, „`c2pa.hash.boxes`“, „`c2pa.hash.collection.data`“, „`c2pa.hash.bmff.v2`“ (veraltet) oder „`c2pa.hash.bmff.v3`“ verweisen.

18.11.2. Schema und Beispiel

Das Schema für diesen Typ wird durch die Regel „`cloud-data-map`” in der folgenden CDDL-Definition definiert:

```
; Assertion, die auf die tatsächlich in der Cloud gespeicherte Assertion verweist
cloud-data-map = {
  „label”: tstr, ; Bezeichnung für die cloudbasierte Behauptung (z. B. c2pa.actions) „size”:
  size-type, ; Anzahl der Datenbytes
  „location”: $hashed-ext-uri-map, ; http(s)-URL, unter der die in der Cloud gehostete Behauptung zu finden ist
  „content_type”: tstr .regexp „^[-\\w.]+/[-+\\w.]+$”, ; Medien-/MIME-Typ für die Daten
  ? „metadata”: $assertion-metadata-map, ; zusätzliche Informationen zur Assertion
}

; Größe ist mindestens 1 in Vielfachen von 1,0
size-type = int .ge 1
```

Ein Beispiel in CBOR-Diagnoseschrift (RFC 8949, Abschnitt 8) ist unten aufgeführt:

```
{
  „size”: 98765,
  „label”: „c2pa.thumbnail.claim”, „location”: {
    "url": "https://some.storage.us/foo",
    "hash": b64'zP84FPSremIrAQHlhw+hRYQdZp/+KggnD0W8opXlIQQ='
  },
  „content_type”: „application/jpeg”
}
```

18.12. Eingebettete Daten

18.12.1. Beschreibung

In früheren Versionen dieser Spezifikation wurde das Konzept einer Datenbox als spezieller Typ einer JUMBF-Box verwendet, um die beliebige Einbettung von Daten in ein C2PA-Manifest zu ermöglichen, beispielsweise für Miniaturansichten, Symbole und inputTo-Inhaltsstoffe. Es wurde festgestellt, dass dies durch einen neuen Box-Typ unnötige Komplexität und fehlende Funktionen mit sich brachte, beispielsweise die Unmöglichkeit, Datenboxen zu redigieren. Daher wurde dieses Konzept zugunsten einer Standardaussage verworfen, die eine Standard-JUMBF-Box vom Typ „Embedded File“ zur Aufnahme der Daten verwendet.

Eine eingebettete Datenaussage muss eine Kennzeichnung haben, die mit `c2pa.embedded-data` beginnt und den Regeln für Aussagekennzeichnungen in Bezug auf mehrere Instanzen entspricht. Darüber hinaus sind einige andere Aussagearten technisch gesehen einer eingebetteten Datenaussage gleichwertig, haben jedoch ihre eigenen eindeutigen Kennzeichnungen (z. B. `c2pa.thumbnail.claim`).

18.12.2. Technische Details

Da die eingebettete Datenaussage auf einem JUMBF-Feld für eingebettete Dateiinhalte basiert, muss ihr Feld „Embedded File Description“ (Beschreibung der eingebetteten Datei) einen IANA-Medientyp (z. B. `image/png`) als Wert für das Feld *„MEDIA TYPE“ (Medientyp)* enthalten und kann einen Dateinamen als Wert für das Feld *„FILE NAME“ (Dateiname*) enthalten. Das `externe` Toggle-Bit darf nicht gesetzt sein.

HINWEIS IANA- -strukturierte Suffixe (https://www.iana.org/assignments/media-type-structured-suffix/media-typstrukturiertes Suffix.xhtml), wie z. B. `+json` und `+zip`, werden ebenfalls als Werte für das Feld *MEDIA TYPE* unterstützt.

Das Feld „Binary Data“ (Binärdaten) der eingebetteten Datenaussage muss die Bits einer Datei (z. B. eines Rasterbildes oder einer Textabfrage) in einem beliebigen Format enthalten, das vom Anspruchgenerator gewünscht wird, aber mit dem im Feld „Embedded File Description“ (Beschreibung der eingebetteten Datei) angegebenen Medientyp übereinstimmt.

18.13. Miniaturansicht

18.13.1. Beschreibung

Eine Miniaturansicht-Angabe liefert eine ungefähre visuelle Darstellung des Assets zu einem bestimmten Zeitpunkt im Lebenszyklus eines Assets. Derzeit gibt es zwei spezifische Ereignisse:

- Import von Inhaltsstoffen und Erstellung von Ansprüchen;
- wobei jedes Ereignis eine eindeutige Bezeichnung für die Assertion verwendet.

18.13.1.1. Miniaturansichten von Ansprüchen

Für thumbnails erstellt unter claim creation time, die thumbnail assertion soll haben die label `c2pa.thumbnail.claim`. In einem C2PA-Manifest darf es nicht mehr als eine Miniaturansicht-Assertion mit diesem Label geben. Frühere

Versionen dieser Spezifikation verlangten, dass der IANA-Registrierungsmedientyp der Miniaturansicht im

Label-Namen enthalten sein (z. B. `c2pa.thumbnail.claim.png`). Diese Namenskonvention ist inzwischen veraltet.

18.13.1.2. Thumbnails von Inhaltsstoffen

Beim Importieren einer Zutat (siehe Abschnitt 10.3.2.2, „Hinzufügen von Zutaten“) sollte auf die im Manifest gespeicherte Miniaturansicht dieser Zutat verwiesen werden. Einige Zutaten enthalten jedoch möglicherweise keine Miniaturansicht oder sogar kein Manifest. In diesem Fall sollte eine neue Miniaturansicht der Zutat generiert und eine neue Miniaturansicht im aktiven Manifest erstellt werden.

Die Miniaturbild-Assertion für eine Zutat muss eine Bezeichnung haben, die mit `c2pa.thumbnail.ingredient` beginnt und den Regeln für Assertion-Bezeichnungen in Bezug auf mehrere Instanzen entspricht. Eine Zutaten-Miniaturansicht könnte beispielsweise die Bezeichnung `c2pa.thumbnail.ingredient 1` haben.

In früheren Versionen dieser Spezifikation musste der IANA-Registrierungsmediatyp der Miniaturansicht im Labelnamen enthalten sein (z. B. `c2pa.thumbnail.claim.png`). Diese Namenskonvention ist inzwischen veraltet.

Frühere Versionen dieser Spezifikation verlangten ein Suffix `_1` für die erste Instanz und einen einzigen Unterstrich. Die aktuelle Spezifikation verwendet durch die Einführung einer einheitlichen Benennung für alle Assertions `c2pa.thumbnail.ingredient` für die erste Instanz, `c2pa.thumbnail.ingredient 1` für die zweite usw. Die bisherige Namenskonvention ist veraltet.

18.13.1.3. Technische Details

Eine Miniaturansicht-Assertion ist eine eingebettete Daten-Assertion, jedoch mit einer speziellen Kennzeichnung, die diesen spezifischen Anwendungsfall identifiziert.

18.14. Aktionen

18.14.1. Beschreibung

Eine Aktionsaussage liefert Informationen zu Bearbeitungen und anderen Aktionen, die sich auf den Inhalt des Assets auswirken. Es gibt eine Reihe von Aktionen – jede Aktion gibt an, *was* mit dem Asset geschehen ist und (optional) *wann* es geschehen ist, zusammen mit möglichen weiteren Informationen, z. B. mit welcher Software die Aktion durchgeführt wurde. Sofern nicht in Abschnitt 18.14.2, „Obligatorisches Vorhandensein mindestens einer Aktionsaussage“, anders angegeben, ist die Reihenfolge der Aktionen in diesem Array nicht festgelegt und gibt nicht die Reihenfolge wieder, in der die Aktionen durchgeführt wurden.

Es gibt zwei Versionen der Aktionsaussage – die ursprüngliche v1 (die die Bezeichnung `c2pa.actions` haben muss) und die neue und verbesserte v2 (die die Bezeichnung `c2pa.actions.v2` haben muss). Aktionen sind nach dem Vorbild von XMP ResourceEvents gestaltet, jedoch mit einer Reihe von C2PA-spezifischen Anpassungen.

v1-Aktionen sind vollständig in ihrem Aktionsarray spezifiziert. In v2 kann eine Aktion jedoch entweder vollständig in einem Element des Aktionsarrays spezifiziert sein oder aus einem Element im Vorlagenarray mit dem gleichen Aktionsnamen abgeleitet werden.

Für jede Aktion, die entweder im Array „`actions`“ oder im Array „`templates`“ vorhanden ist, muss der Wert des Aktionsfeldes entweder ein vordefinierter Aktions Name (`c2pa.resized`, `c2pa.edited`, usw.) oder entitätsspezifischer Aktions Name

(`com.fabrikam.gaussianBlur` usw.).

Die vordefinierten Namen mit dem Präfix „`c2pa.`“ sind in Tabelle 8 „Liste der vordefinierten Aktionen“ aufgeführt:

Tabelle 8. Liste vordefinierter Aktionen

Aktion	Bedeutung
c2pa.addedText	(sichtbar) Textinhalt wurde in das Asset eingefügt, z. B. auf einer Textebene oder als Bildunterschrift.
c2pa.adjustedColor	Änderungen an Farbton, Sättigung usw.
c2pa.changedSpeed	Reduzierte oder erhöhte Wiedergabegeschwindigkeit einer Video- oder Audiospur
c2pa.color_adjustments	[VERALTET] Änderungen an Ton, Sättigung usw.
c2pa.converted	Das Format der Ressource wurde geändert.
c2pa.created	Das Asset wurde zum ersten Mal erstellt.
c2pa.cropped	Teile des digitalen Inhalts des Vermögenswerts wurden ausgeschnitten.
c2pa.deleted	Teile des digitalen Inhalts des Assets wurden gelöscht.
c2pa.drawing	Änderungen mit Zeichenwerkzeugen wie Pinseln oder Radiergummi.
c2pa.dubbed	Es wurden Änderungen an Audio vorgenommen, in der Regel an einer oder mehreren Spuren eines zusammengesetzten Assets.
c2pa.edited	Allgemeine Aktionen, die als redaktionelle Veränderungen des Inhalts angesehen werden können.
c2pa.edited.metadata	Änderungen an den Metadaten eines Assets oder einer Metadatenangabe, jedoch nicht am digitalen Inhalt des Assets.
c2pa.enhanced	Angewandte Verbesserungen wie Rauschunterdrückung, Mehrbandkompression oder Schärfung, die keine redaktionellen Veränderungen des Inhalts darstellen.
c2pa.filtered	Änderungen am Erscheinungsbild durch angewendete Filter, Stile usw.
c2pa.opened	Ein vorhandenes Asset wurde geöffnet und wird als `parentOf` Zutat festgelegt.
c2pa.orientation	Änderungen an der Ausrichtung und Position von Inhalten.
c2pa.placed	Eine oder mehrere `Komponenten` der Inhaltsstoffe wurden dem Asset hinzugefügt/zugeordnet.
c2pa.veröffentlicht	Asset wird für ein breiteres Publikum freigegeben.
c2pa.redacted	Eine oder mehrere Aussagen wurden redigiert.
c2pa.removed	`Eine Komponente` der Zutat wurde entfernt.

c2pa.repackaged	Eine Umwandlung eines Verpackungs- oder Behälterformats in ein anderes. Der Inhalt wird ohne Transcodierung neu verpackt. Diese Aktion wird als eine nicht redaktionelle Umwandlung des `parentOf`-Bestandteils betrachtet.
c2pa.resized	Änderungen an den Abmessungen des Inhalts, seiner Dateigröße oder beidem
c2pa.transcoded	Eine Konvertierung von einer Kodierung in eine andere, einschließlich Auflösungsskalierung, Bitratenanpassung und Änderung des Kodierungsformats. Diese Aktion wird als nicht-redaktionelle Transformation der `parent`Of-Komponente betrachtet.
c2pa.translated	Änderungen an der Sprache des Inhalts.
c2pa.trimmed	Entfernen eines zeitlichen Bereichs des Inhalts.
c2pa.unknown	Es ist etwas passiert, aber der claim_generator kann nicht angeben, was genau.
c2pa.watermarked	In den digitalen Inhalt wurde ein unsichtbares Wasserzeichen eingefügt, um eine weiche Bindung zu schaffen.

Darüber hinaus wird die folgende Reihe vordefinierter Namen (in Tabelle 9, „Liste der Schriftartenaktionen“), denen das Präfix `font.` vorangestellt ist, speziell für Schriftarten-Assets verwendet:

HINWEIS In einer früheren Version dieser Spezifikation wurden diese als `c2pa.font` bezeichnet, was jedoch zugunsten des kürzeren Präfixes `font` verworfen wurde.

Tabelle 9. Liste der Schriftaktionen

Aktion	Bedeutung
font.charactersAdded	Hinzugefügte Zeichen oder Zeichensätze.
font.charactersDeleted	Gelöschte Zeichen oder Zeichensätze.
font.charactersModified	Hinzugefügte und gelöschte Zeichen oder Zeichensätze.
font.createdFromVariableFont	Die Schriftart wurde ganz oder teilweise aus einer variablen Schriftart instanziiert.
font.edited	Die Schriftart wurde einer Bearbeitungsaktion unterzogen, die nicht durch eine spezifischere Aktion beschrieben wird.
font.hinted	Hinting angewendet.
font.merged	Die Schriftart ist eine Kombination aus früheren Schriftarten.
font.openTypeFeatureAdded	OpenType-Funktion zur Schriftart hinzugefügt.
font.openTypeFeatureModified	OpenType-Funktion geändert.
font.openTypeFeatureRemoved	OpenType-Funktion aus Schriftart entfernt.
font.subset	Die Schriftart wurde reduziert, um eine beliebige (sui generis) Untergruppe von Zeichen zu unterstützen.

18.14.2. Mindestens eine Aktionsaussage muss vorhanden sein.

Es muss sein mindestens mindestens eine Handlungen Aussage vorhanden in entweder der `created_assertions` oder `gathered_assertions`-Array des Anspruchs eines Standard-C2PA-Manifests. Darüber hinaus:

- Wenn das Asset *neu* erstellt wurde (z. B. durch Ausführen der Option „Datei → Neu“ in einem Kreativtool, Aufnehmen eines Fotos oder Videos oder Generieren des Mediums durch ein generatives KI-Modell), muss das Array „actions“ in der ersten c2pa.actions-Assertion entweder im Array „created_assertions“ oder „gathered_assertions“ des Anspruchs als erstes Element eine c2pa.created-Aktion enthalten.
    - Für alle Assets muss ein entsprechendes Feld „digitalSourceType“ mit einem geeigneten Wert zusammen mit der Aktion „c2pa.created“ aufgezeichnet werden, um die Art des Assets bei seiner Entstehung anzugeben. Wenn das Asset ohne digitalen Inhalt erstellt wird, muss das Feld „digitalSourceType“ den Wert „http://c2pa.org/digitalsourcetype/empty“ haben.
- Wenn das Asset durch Öffnen eines vorhandenen Assets als parentOf-Komponente zur Bearbeitung erstellt wurde, muss das Aktionsarray in der ersten c2pa.actions-Assertion entweder im Array `created_assertions` oder im Array `gathered_assertions` des Anspruchs als erstes Element eine c2pa.opened-Aktion enthalten. In Verbindung mit einer c2pa.opened-Aktion ist kein digitalSourceType-Feld erforderlich.

HINWEIS Diese Anforderung gilt nicht für Update Manifests.

HINWEIS Beachten Sie bei der Aufzeichnung von Aktionen in `gathered_assertions`, dass diese Assertions nicht dem Unterzeichner zugeordnet werden (siehe Kapitel 10, *Claims*).

Der vollständige Satz von Aktionsaussagen in einem C2PA-Manifest darf nicht mehr als eine Aktion enthalten, deren Typ entweder `c2pa.created` oder `c2pa.opened` ist. Wenn eine dieser Aktionen in `created_assertions` erscheint, darf keine der beiden in `gathered_assertions` erscheinen, und wenn eine in `gathered_assertions` erscheint, darf keine der beiden in `created_assertions` erscheinen.

BEISPIEL: Ein generatives KI-Modell generiert als Reaktion auf eine Textanweisung ein Video. Das aktive Manifest des resultierenden Video-Assets würde eine c2pa.actions-Assertion enthalten, die mit einer c2pa.created-Aktion beginnt, die selbst einen Wert von `http://cv.iptc.org/newscodes/digitalsourcetype/trainedAlgorithmicMedia` im entsprechenden Feld `digitalSourceType`.

BEISPIEL: Ein Benutzer öffnet Emily's Mobile Poster Maker, um ein Bild für einen Social-Media-Beitrag zu erstellen. Der Benutzer wählt eine Vorlage aus und beginnt dann mit der Anpassung, wobei er einige vorhandene Fotos importiert. Das aktive Manifest des resultierenden Bild-Assets würde eine c2pa.actions-Assertion enthalten, die mit einer c2pa.created-Aktion beginnt und kein digitalSourceType-Feld enthält, was darauf hinweist, dass dies als neue Datei begonnen hat. Es würde auch eine c2pa.placed-Aktion für jedes vom Benutzer importierte Foto enthalten, die jeweils auf eine entsprechende Ingredient-Assertion verweist, in der eine componentOf-Beziehung angegeben ist. Schließlich werden zusätzliche Aktionen für andere vom Benutzer durchgeführte Vorgänge aufgezeichnet.

BEISPIEL: Die Medienredaktion einer Zeitung möchte ein Foto bearbeiten, das von einem Fotojournalisten mit einer C2PA-fähigen Kamera aufgenommen wurde. Der Medienredakteur öffnet das Foto und wendet Zuschneide- und Vignettierungsfunktionen an. Das aktive Manifest des bearbeiteten Foto-Assets enthält eine c2pa.actions-Assertion mit einer c2pa.opened-Aktion, die auf eine Ingredient-Assertion für das Originalfoto verweist, wobei eine parentOf-Beziehung angegeben ist. Es würde auch Aktionen für die

Zuschneide- und Vignettenbearbeitungen enthalten.

18.14.3. Alle enthaltenen Aktionen

Die `actions-map-v2` kann ein Feld namens `allActionsIncluded` enthalten, das einen booleschen Wert hat. Wenn `allActionsIncluded` vorhanden ist und den Wert „true“ hat, gibt der Anspruchgenerator an, dass nur die in der actions-Assertion aufgeführten Aktionen für das Asset durchgeführt wurden. Wenn `allActionsIncluded` nicht vorhanden ist oder den Wert „false“ hat, kann ein Manifest-Consumer davon ausgehen, dass andere Aktionen durchgeführt wurden, die jedoch nicht aufgeführt sind.

18.14.4. Felder in der actions-Assertion

18.14.4.1. Beschreibung

Eine Aktion kann im Feld `description` eine Freitextbeschreibung der Aktion enthalten. Dies ist besonders nützlich für nicht standardmäßige Aktionen, kann jedoch auch verwendet werden, um zusätzliche Informationen zu einer Standardaktion bereitzustellen. Beispielsweise könnte eine Aktion `c2pa.edited` die `Beschreibung` „Paintbrush tool” (Pinsel-Werkzeug) haben.

18.14.4.2. Grund

Wenn vorhanden, muss das Feld „`Grund`“ einen der folgenden Standardwerte oder einen benutzerdefinierten Wert enthalten, der derselben Syntax wie entitätsspezifische Namespaces entspricht, um die Begründung für die Aktion anzugeben:

- `c2pa.PII.present`;
- `c2pa.invalid.data`;
- `c2pa.trade-secret.present`;
- `c2pa.government.confidential`.

HINWEIS	Obwohl das Feld „`Grund`” für alle Aktionen verwendet werden kann, sind derzeit nur auf die Schwärzung ausgerichtete c2pa-Werte definiert.

Bei Verwendung einer `c2pa.redacted`-Aktion muss das Feld „`Grund`“ die Begründung für die Redigierung enthalten. Weitere Anforderungen für die `c2pa.redacted`-Aktion finden Sie in Abschnitt 18.14.4.7, „Parameter“.

18.14.4.3. Wenn

Möglicherweise sind auch das Datum und die Uhrzeit, zu denen die Aktion stattgefunden hat, im Feld „`when`“ angegeben. Wenn ja, muss der Wert des Feldes „`when`“
mit den CBOR-Datums-/Zeitangaben (RFC 8949, 3.4.1) übereinstimmen.

HINWEIS	Das Feld „`when`“ dient als einfacher, nicht vertrauenswürdiger Zeitstempel. Es werden UTC-basierte Zeiten empfohlen.

18.14.4.4. SoftwareAgent

Die zur Ausführung der Aktion verwendete Software oder Hardware kann über das Feld „`softwareAgent`“ identifiziert werden. In einer v1-Aktion handelt es sich dabei um eine einfache Textzeichenfolge. In v2 verwendet „`softwareAgent`“ jedoch die umfangreichere Struktur „`generator-info-map`“ als

beschrieben in Abschnitt 10.2.3.2, „Generator-Info-Map“. Wenn mehrere Software-Agenten verwendet werden, wie in Abschnitt 18.14.6.2, „Software-Agenten“, beschrieben, dann muss das Feld „softwareAgentIndex“ verwendet werden, um den Software-Agenten anhand seines 0-basierten Index im Array „softwareAgents“ zu referenzieren. Eine bestimmte Aktion darf nur ein Feld „softwareAgent“ oder „softwareAgentIndex“ haben.

HINWEIS

Diese Felder sind nützlich, wenn der SoftwareAgent nicht dasselbe Programm wie der Anspruchgenerator ist.

HINWEIS

Eine frühere Version dieser Spezifikation enthielt auch ein Feld „Actors“, das jedoch in Version 2.0 entfernt wurde.

18.14.4.5. Typ der digitalen Quelle

Eine Aktion kann einen Schlüssel „digitalSourceType“ enthalten, dessen Wert einer der von der IPTC definierten Begriffe oder ein C2PA-spezifischer Wert aus der folgenden Liste sein muss:

http://c2pa.org/digitalsourcetype/empty

Medien, deren digitale Inhalte praktisch leer sind, wie beispielsweise eine leere Leinwand oder ein Video ohne Länge.

http://c2pa.org/digitalsourcetype/trainedAlgorithmicData

Daten, die das Ergebnis der algorithmischen Verwendung eines Modells sind, das aus Stichproben von Inhalten und Daten abgeleitet wurde. Unterscheidet sich von „http://cv.iptc.org/newscodes/digitalsourcetype/trainedAlgorithmicMedia“ dadurch, dass das Ergebnis kein Medientyp (z. B. Bild oder Video) ist, sondern ein Datenformat (z. B. CSV, Pickle).

HINWEIS

Eine common use case for the digitalSourceType key is in conjunction with the c2pa.created Aktion, um anzugeben, wie das Medienelement erstellt wurde – beispielsweise „digitale Erfassung“, „aus Negativ digitalisiert“ oder „trainierte algorithmische Medien“.

Bei „trainierten algorithmischen“ Assets und Daten, wie sie beispielsweise durch generative KI erstellt werden, können dem C2PA-Manifest eine oder mehrere Komponenten hinzugefügt werden, um Informationen über die Eingaben zu liefern, die zur Erstellung des Assets geführt haben. Sie können aus einer c2pa.placed- oder c2pa.created-Aktion referenziert werden, wie in Beispiel 8, „Beispiel für eine Aktion für generative KI“, gezeigt.

18.14.4.6. Änderungen

Die Aktion kann sich nur auf einen Teil eines Assets beziehen – beispielsweise auf einen Bereich von Frames in einem Video oder einen bestimmten Bereich eines Bildes. In Version 1 war der Wert eine einfache Textzeichenfolge. In Version 2 werden sie anhand eines Änderungsfeldes identifiziert, dessen Wert ein Array von Region-Map-Objekten ist (wie in Abschnitt 18.2, „Regions of Interest“, definiert).

18.14.4.7. Parameter

Eine Aktion kann einen Parameterschlüssel enthalten, der die Angabe einiger aktionsspezifischer Informationen über vordefinierte sowie die offene Einbindung beliebiger benutzerdefinierter Felder (und der damit verbundenen Werte) ermöglicht. Benutzerdefinierte Felder müssen derselben Syntax entsprechen wie entitätsspezifische Namespaces, z. B. com.litware.someFieldName.

HINWEIS

Dies ist nützlich, um zusätzliche Informationen bereitzustellen, die für einen bestimmten Workflow oder C2PA Manifest Consumer hilfreich sind.

Ein Claim-Generator, der dieselbe Aktion mit denselben Parametern und Einstellungen wiederholt ausführt, kann das Feld „multipleInstances“ verwenden, um anzugeben, ob die Aktion mehrfach ausgeführt wurde oder nicht. Ist das Feld „multipleInstances“ nicht vorhanden, ist nicht bekannt, ob die Aktion mehrfach ausgeführt wurde.

Bei Verwendung einer c2pa.opened- oder c2pa.placed-Aktion muss das Feld „ingredient“ (für v1) oder „ingredients“ (für v2) im Parameterobjekt die gehashten JUMBF-URIs zu einer oder mehreren zugehörigen Inhaltsstoffaussagen enthalten. Bei einer c2pa.removed-Aktion muss dieses Feld die gehashte JUMBF-URI zu einer componentOf-Inhaltsstoffaussage in einem anderen Manifest enthalten. In einigen Fällen ist nur ein Teil eines Inhaltsstoffs für die Aktion relevant. In solchen Fällen sollte die Inhaltsstoffaussage Metadaten enthalten, die ein Feld „regionOfInterest“ enthalten, das zur Angabe der relevanten Regionen des Inhaltsstoffs verwendet wird (wie in Abschnitt 18.15.13, „Metadaten zu Inhaltsstoffen“, beschrieben).

HINWEIS

In früheren Versionen dieser Spezifikation wurden die Aktionen „c2pa.transcoded“ und „c2pa.repackaged“ mussten auf die parentOf-Ingredient-Assertion verweisen, auf die die vorhergehende c2pa.opened-Aktion referenzierte „parentOf ingredient“-Assertion; Claim-Generatoren können dies aus Kompatibilitätsgründen mit älteren Validatoren tun.

Bei Verwendung einer c2pa.translated-Aktion müssen die Felder sourceLanguage und targetLanguage im Parameterobjekt die Sprachcodes gemäß RFC 5646, BCP 47 enthalten.

Beispiel 8. Beispiel für eine Aktion für generative KI

Die Aktion „c2pa.created” für ein Bild, das von einem generativen KI-Modell erstellt wurde, könnte in der CBOR-Diagnoseschreibweise (RFC 8949, Abschnitt 8) wie folgt aussehen:

```
// Eine Aktionsaussage zur Beschreibung der Ausgabe generativer KI //
{
  „actions”: [
     {
       „action”: „c2pa.created”,

       „softwareAgent“: {
            „name“: „Joe's Photo Editor“, „version“: „2.0“,
            „operating_system“: „Windows 10“
       },
       „digitalSourceType“: „http://cv.iptc.org/newscodes/digitalsourcetype/trainedAlgorithmicMedia“,
       „parameters“: { „ingredients“: [
            {
              „url“: „self#jumbf=c2pa.assertions/c2pa.ingredient.v3“, „alg“:
              „sha256“,
              „hash”: b64'...',
            },
            {
              „url”: „self#jumbf=c2pa.assertions/c2pa.ingredient.v3 1”, „alg”:
              „sha256”,
              „hash“: b64'...',
            }
         ]
```

```
       }
     }
  ]
}
```

Bei Verwendung einer c2pa.redacted-Aktion muss das Feld „redacted" im Parameterobjekt die JUMBF-URI zu der redigierten Assertion enthalten.

18.14.5. Wasserzeichen

Bei Verwendung einer c2pa.watermarked-Aktion muss auch eine Soft-Binding-Assertion in das C2PA-Manifest aufgenommen werden, um das eingefügte Wasserzeichen zu beschreiben.

18.14.6. Aktionsvorlagen

18.14.6.1. Vorlagen

Die Elemente des Vorlagen-Arrays in einer v2-Aktion werden anhand einer Kombination aus allgemeinen Elementen zu Aktionen und einigen vorlagenspezifischen Werten beschrieben. Diese Werte werden von einem C2PA-Manifest-Consumer mit Aktionen mit demselben Namen oder mit allen Aktionen (wenn der Wert des Aktionsfeldes der spezielle Wert „ " ist), um ein vollständiges Bild einer Aktion zu erhalten. Wenn mehrere Vorlagen für dieselbe Aktion gelten, werden die Werte beginnend mit der Vorlage (falls vorhanden) zusammengeführt und dann in der Reihenfolge angewendet, in der sie im Vorlagen-Array erscheinen.

Beispiel 9. Beispiel für eine Aktionsvorlage

Eine Aktion und eine Vorlage in CBOR-Diagnoseschrift (RFC 8949, Abschnitt 8):

```
// Beispiel für eine einzelne Vorlage, die auf mehrere Aktionen angewendet wird //
{
     "actions": [
          {
               „action“: „com.joesphoto.filter“, „when“:
               0(„2020-02-11T09:00:00Z“)
          },
          {
               „action”: „c2pa.edited”,

          },
          {
               „action”: „com.joesphoto.filter”, „when”:
               0(„2020-02-11T09:20:00Z”)
          },
          {
               „action”: „c2pa.cropped”,

          }
     ],
     „templates”: [{
          „action“: „com.joesphoto.filter“, „description“:
          „Magic Filter“,
```

```
          „digitalSourceType“: „http://cv.iptc.org/newscodes/digitalsourcetype/compositeSynthetic“,
          „softwareAgent“: {
               „name“: „Joe's Photo Editor“, „version“:
               „2.0“,
               „operating_system“: „Windows 10“
          }
     }]
}

// Beispiel für die Verwendung der speziellen Vorlage „all actions/`*`“ für alle Aktionen //
{
     "actions": [
          {
               „action”: „c2pa.created”, „when”:
               0(„2024-03-09T20:04Z”)
          },
          {
               „action”: „c2pa.edited”,

          },
          {
               „action”: „c2pa.cropped”,

          }
     ],
     „templates”: [{
          „action“: „*“,
          „digitalSourceType“:
„http://cv.iptc.org/newscodes/digitalsourcetype/humanEdits“,
          „softwareAgent“ : {
               „name“: „Jane's Human Authoring Tool“, „version“: „1.0“
          }
     }]
}
```

Ein C2PA-Manifest-Verbraucher muss die Werte aus der Vorlage übernehmen und mit den Werten aus der Aktion selbst überlagern, wodurch alle Werte mit demselben Namen ersetzt werden.

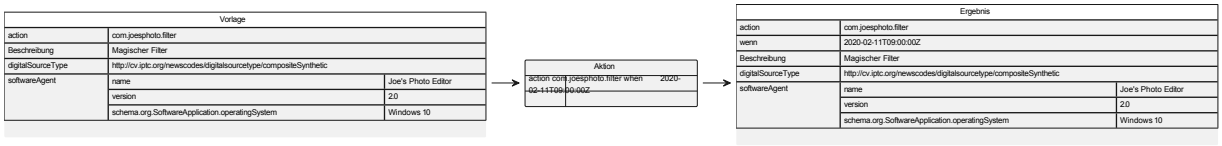

Abbildung 18. Ablauf der Aktionsvorlage

Eine Vorlage kann einen Schlüssel „`templateParameters`“ enthalten, der die Einbindung beliebiger anderer Schlüssel (und der zugehörigen Werte) ermöglicht. Dies ist nützlich, um zusätzliche Informationen bereitzustellen, die für einen bestimmten Workflow oder C2PA Manifest Consumer hilfreich sind.

18.14.6.2. SoftwareAgents

Wenn mehrere SoftwareAgents verwendet wurden, können diese stattdessen im Feld `softwareAgents` aufgelistet werden. Dieses Feld ist ein Array von generator-info-map-Objekten, von denen jedes eine andere Software oder Hardware beschreibt, auf die dann über den Index im Feld `softwareAgentIndex` einer bestimmten Aktion oder Vorlage verwiesen werden kann.

Beispiel 10. Beispiel für Software-Agenten

Ein Beispiel für die Angabe mehrerer Agenten über mehrere Aktionen hinweg in CBOR-Diagnoseschrift (RFC 8949, Abschnitt 8):

```
{
     „actions”: [
          {
               „action”: „com.joesphoto.magic-avatar”, „when”:
               0(„2020-02-11T09:00:00Z”),
               „softwareAgentIndex” : 0
          },
          {
               „action”: „c2pa.edited”,

               „softwareAgentIndex“: 1
          },
          {
               „action”: „com.joesphoto.beauty-filter”, „when”:
               0(„2020-02-11T09:20:00Z”),
               „softwareAgentIndex“: 0
          },
          {
               „action“: „com.joesphoto.all-smiles“, „when“: 0
               („2020-02-11T09:40:00Z“),
               „softwareAgentIndex“: 0
          },
          {
               „action”: „c2pa.cropped”,

               „softwareAgentIndex“: 1
          },
          {
               „action”: „com.joesphoto.green-screen”, „when”:
               0(„2020-02-11T09:50:00Z”),
               „softwareAgentIndex“: 0
          }
     ],
     „Softwareagenten“: [
          {
               „name“: „Joe's AI Filter“,
               „version”: „1.0”,
               „operating_system“: „Windows 10“
          }
          {
               „name“: „Joe's Photo Editor“,
               „version“: „2.0“,
               „Betriebssystem“: „Windows 10“
          }
     ]
}
```

18.14.6.3. Symbole

Eine Vorlage kann auch ein Symbol enthalten – ein Bild (Raster oder Vektor), das in der Benutzererfahrung des C2PA Manifest Consumers verwendet werden kann, um die Aktion grafisch darzustellen. Da ein Manifest Consumer über alle

Die definierten Aktionen, solche Symbole dürfen nur in Vorlagen für entitätsspezifische Aktionen vorhanden sein.

Der Wert des Symbolfelds muss, sofern vorhanden, eine gehasht URI sein. Diese gehasht URI muss entweder zu einer eingebetteten Datenaussage oder zu einer Cloud-Datenaussage führen. Wenn eine eingebettete Datenaussage verwendet wird, muss ihre Bezeichnung `c2pa.icon` lauten und den Regeln für Aussagebezeichnungen in Bezug auf mehrere Instanzen entsprechen.

HINWEIS Dieses Symbolfeld ist in seiner Struktur identisch mit dem Symbolfeld in der Generator-Info-Map des Anspruchs.

Manifest-Konsumenten sollten auch den in früheren Versionen dieser Spezifikation empfohlenen Datenbox-Ansatz unterstützen.

18.14.7. Lokalisierungen

Wenn die `Metadaten` einer Aktionsaussage ein Lokalisierungswörterbuch für eine Vorlage enthalten, gelten die Lokalisierungen auch für alle Aktionen, die auf dieser Vorlage basieren.

18.14.8. Verwandte Aktionen

Wenn eine Reihe von Aktionen miteinander in Zusammenhang stehen und in der Regel gleichzeitig stattfinden, kann es sinnvoll sein, sie entsprechend zu verknüpfen. Das Feld „`Related`“ (`Verwandte`) in der v2-Aktion bietet Platz für die Auflistung der zusätzlichen Aktionen, die miteinander in Zusammenhang stehen. Jede verwandte Aktion sollte eine Untergruppe der primären Aktion sein und nur die Felder enthalten, die sich unterscheiden. Genau wie bei einer Aktionsvorlage werden die Werte von einem C2PA Manifest Consumer mit denen der primären Aktion zusammengeführt, um ein vollständiges Bild jeder verbundenen Aktion zu erhalten.

18.14.9. Asset-Wiedergaben

Asset-Renderings sind bei der Verbreitung von Medien im Internet weit verbreitet. Diese Renderings werden häufig erstellt, um Medien an Verbraucher mit unterschiedlichen Verbindungsgeschwindigkeiten, Bildschirmauflösungen und anderen Umgebungen zu liefern. Wir können die Aktionsaussage verwenden, um den konsumierenden Akteuren zu helfen, die Absicht bestimmter Anspruchsteller zu verstehen, Asset-Renderings zu erstellen.

Das Vorhandensein nur der Aktionen `c2pa.published`, `c2pa.transcoded` und `c2pa.repackaged` in einer `c2pa.`actions-Assertion signalisiert dem Manifest-Verbraucher, dass der Unterzeichner versichert, dass zwischen den Inhaltskomponenten und dieser Komponente keine redaktionellen Änderungen an den digitalen Inhalten vorgenommen wurden.

Das zusätzliche Vorhandensein eines einzelnen „parentOf“-Inhaltselements signalisiert dem Manifest-Verbraucher zusätzlich, dass der Unterzeichner versichert, dass das Asset direkt von diesem übergeordneten Element abgeleitet wurde.

18.14.10. Soft Binding Lookup

Bei der Ausführung einer `c2pa.`opened- oder `c2pa.`placed-Aktion mit einem Asset, das kein C2PA-Manifest enthält, kann der Anspruchgenerator eine Soft-Binding-Suche verwenden, um das C2PA-Manifest für dieses Asset zu finden. Bei Erfolg sollte der Anspruchgenerator das gefundene C2PA-Manifest als Wert des Feldes `activeManifest` in der Ingredient Assertion hinzufügen. In diesem Fall muss die Ingredient Assertion auch ein Feld `„softBindingsMatched”` mit dem Wert „true” und ein `Feld „softBindingAlgorithmsMatched”` enthalten, dessen Wert mindestens einen Eintrag im Array enthält.

HINWEIS	Das Hinzufügen dieser Felder zeigt dem C2PA-Manifest-Verbraucher an, dass eine Soft-Binding-Suche verwendet wurde.
HINWEIS	Da die meisten wiederhergestellten Soft-Binding-Manifeste eine Hard-Binding-Assertion enthalten, die nicht mit dem gesuchten Asset übereinstimmt, ist zu erwarten, dass Validierungsfehler in der Ingredient Assertion gemeldet werden.

Ein Beispiel für eine Zutatenaktion, die zeigt, dass ihr Manifest über Soft-Binding abgerufen wurde, in CBOR-Diagnoseschrift (RFC 8949, Abschnitt 8):

```
// Eine Zutat-Assertion, deren Manifest über Soft-Binding abgerufen wurde //
{
  „dc:title”: „image 1.jpg”, „dc:format”:
  „image/jpeg”, „relationship”: „parentOf”,
  „softBindingsMatched”: true,
  „softBindingAlgorithmsMatched”: [
     „com.foo.watermark.1”
  ]
  „activeManifest”: {
       "url": "self#jumbf=/c2pa/urn:c2pa:5E7B01FC-4932-4BAB-AB32-D4F12A8AA322", "hash":
       b64'1kjJTO108b71cL95UxgfHD3eDgk9VrCedW8n3fYTRMk='
  },
}



// eine Aktionsaussage, die auf die Zutat verweist //
{
  „actions”: [
     {
       „action”: „c2pa.opened”,

       „softwareAgent”: {
          „name“: „Joe's Photo Editor“, „version“:
          „2.0“,
          „operating_system“: „Windows 10“
       },
       „Parameter“: {
          „Inhaltsstoffe“: [
             {
               „URL“: „self#jumbf=c2pa.assertions/c2pa.ingredient.v3“, „Alg“:
               „sha256“,
               „hash“: „b64'...“
             }
          ]
       }
     }
  ]
}
```

18.14.11. Veraltete Aktionen

Die folgenden Aktionen waren Teil früherer Versionen dieser Spezifikation und sind inzwischen veraltet:

- `c2pa.copied`;

- `c2pa.formatted`;
- `c2pa.version_updated`;
- `c2pa.printed`;
- `c2pa.managed`;
- `c2pa.produziert`;
- `c2pa.gespeichert`.

Sie werden nicht mehr in `c2pa.actions-` oder `c2pa.actions.`v2-Assertions geschrieben, können jedoch in bereits vorhandenen C2PA-Manifesten erscheinen.

18.14.12. Schema und Beispiel

Das Schema für `c2pa.actions` wird durch die Regel `actions-map` definiert, und das Schema für `c2pa.actions.v2` wird durch die Regel `actions-map-v2` in der folgenden CDDL-Definition definiert:

CDDL für Aktionen

```
actions-map = {
  „actions” : [1* action-items-map], ; Liste der Aktionen
  ? „Metadaten“: $assertion-metadata-map, ; zusätzliche Informationen zur Assertion
}

$action-choice /= „c2pa.addedText”
$action-choice /= „c2pa.adjustedColor”
$action-choice /= „c2pa.changedSpeed”
$action-choice /= „c2pa.color_adjustments”
$action-choice /= „c2pa.converted”
$action-choice /= „c2pa.copied”
$action-choice /= „c2pa.created”
$action-choice /= „c2pa.cropped”
$action-choice /= „c2pa.deleted”
$action-choice /= „c2pa.drawing”
$action-choice /= „c2pa.dubbed”
$action-choice /= „c2pa.bearbeitet”
$action-choice /= „c2pa.bearbeitet.Metadaten”
$action-choice /= „c2pa.gefiltert”
$action-choice /= „c2pa.formatiert”
$action-choice /= „c2pa.verwaltet”
$action-choice /= „c2pa.opened”
$action-choice /= „c2pa.orientation”
$action-choice /= „c2pa.produced”
$action-choice /= „c2pa.placed”
$action-choice /= „c2pa.gedruckt”
$action-choice /= „c2pa.veröffentlicht”
$action-choice /= „c2pa.redacted”
$action-choice /= "c2pa.removed"
$action-choice /= „c2pa.repackaged”
$action-choice /= „c2pa.resized”
$action-choice /= „c2pa.saved”
$action-choice /= „c2pa.transcoded”
$action-choice /= „c2pa.translated”
$action-choice /= „c2pa.trimmed”
$action-choice /= „c2pa.unknown”
```

```
$action-choice /= „c2pa.version_updated”
$action-choice /= „c2pa.watermarked”
$action-choice /= „font.edited”
$action-choice /= „font.subset”
$action-choice /= „font.createdFromVariableFont”
$action-choice /= „font.charactersAdded”
$action-choice /= „font.charactersDeleted”
$action-choice /= „font.charactersModified”
$action-choice /= „font.hinted”
$action-choice /= „font.openTypeFeatureAdded”
$action-choice /= „font.openTypeFeatureModified”
$action-choice /= „font.openTypeFeatureRemoved”
$action-choice /= „font.merged”
$action-choice /= tstr .regexp "([\\da-zA-Z_-]+\\.)+[\\da-zA-Z_-]+" buuid =

#6.37(bstr)

; HINWEIS: Eine frühere Version dieser Spezifikation enthielt auch ein Feld „Akteure“, das jedoch in Version
2.0 entfernt wurde.
action-items-map = { „action”:
  $action-choice,
  ? „when”: tdate, ; Zeitstempel, zu dem die Aktion stattfand.
  ? „softwareAgent”: tstr .size (1..max-tstr-length), ;Der Software-Agent, der die Aktion ausgeführt hat.
  ? „changed”: tstr .size (1..max-tstr-length), ; Eine durch Semikolons getrennte Liste der Teile der
Ressource, die seit dem letzten Ereignis geändert wurden. Wenn nicht vorhanden, wird davon ausgegangen, dass
sie undefiniert ist. Wenn Änderungen verfolgt werden und der Umfang der geänderten Komponenten unbekannt ist,
kann davon ausgegangen werden, dass sich alles geändert haben könnte.
  ? „instanceID”: buuid, ; Der Wert der Eigenschaft xmpMM:InstanceID für die geänderte (Ausgabe-)Ressource
  ? „parameters”: parameters-map, ; Zusätzliche Parameter der Aktion. Diese variieren häufig je nach Art der
Aktion
  ? „digitalSourceType”: tstr .size (1..max-tstr-length), ; Einer der definierten Quelltypen unter
https://cv.iptc.org/newscodes/digitalsourcetype/
}

parameters-map = {
  ? „ingredient”: $hashed-uri-map, ; Eine Hash-URI zur Ingredient-Assertion, auf die diese Aktion wirkt
  ? „description”: tstr .size (1..max-tstr-length) ; Zusätzliche Beschreibung der Aktion
  * tstr => any
}


; Version 2 (v2) der Aktionsaussage

$action-reason /= "c2pa.PII.present"
$action-reason /= „c2pa.invalid.data”
$action-reason /= „c2pa.tradesecret.present”
$action-reason /= „c2pa.government.confidential”
$action-reason /= tstr .regexp "([\\da-zA-Z_-]+\\.)+[\\da-zA-Z_-]+"

actions-map-v2 = {
  "actions" : [1* action-item-map-v2],              ; Liste der Aktionen
  ? "templates": [1* $action-template-map-v2], ; Liste der Vorlagen für die Aktionen
  ? „softwareAgents”: [1* $generator-info-map], ; Liste der Software/Hardware, die die Aktion ausgeführt hat
  ? „metadata”: $assertion-metadata-map,              ; zusätzliche Informationen zur Assertion
  ? „allActionsIncluded”: bool                        ; Wenn vorhanden und wahr, bedeutet dies, dass keine
Aktionen stattgefunden haben, die nicht in der Aktionsliste enthalten waren.
}

action-common-map-v2 = {
```

```
  ? „softwareAgent”: $generator-info-map, ; Beschreibung der Software/Hardware, die die Aktion ausgeführt hat
  ? „softwareAgentIndex”: int, ; 0-basierter Index in das softwareAgents-Array in actions-map-2
  ? „description”: tstr .size (1..max-tstr-length), ; Zusätzliche Beschreibung der Aktion, wichtig für
benutzerdefinierte Aktionen
  ? „digitalSourceType”: tstr .size (1..max-tstr-length), ; Einer der definierten Quelltypen unter
https://cv.iptc.org/newscodes/digitalsourcetype/ oder in dieser Spezifikation
}

; HINWEIS: Eine frühere Version dieser Spezifikation enthielt auch ein Feld „Akteure“, das jedoch in Version
2.0 entfernt wurde.
action-item-map-v2 = {
  „action”: $action-choice , ; Art der Aktion action-common-
  map-v2, ; jetzt zusätzliche allgemeine Elemente
  ? „when”: tdate, ; Zeitstempel, wann die Aktion stattgefunden hat.
  ? „changes”: [1* region-map], ; Eine Liste der Regionen von Interesse der Ressource, die geändert wurden.
Wenn nicht vorhanden, wird davon ausgegangen, dass sie undefiniert sind.
  ? „related”: [1* action-item-map-v2], ; Liste der zugehörigen Aktionen
  ? „reason“: $action-reason, ; der Grund, warum diese Aktion durchgeführt wurde, erforderlich, wenn die Aktion
`c2pa.redacted` ist
  ? „parameters”: parameters-map-v2 ; Zusätzliche Parameter der Aktion. Diese variieren häufig je nach Art der
Aktion
}

action-template-map-v2 = {
  „action”: $action-choice / „*”, ; Vorlagen unterstützen die zusätzliche Sonderoption „*” action-common-
  map-v2,                          ; zusätzliche allgemeine Elemente
  ? „icon”: $hashed-uri-map, ; hashed_uri-Verweis auf eine eingebettete Datenaussage
  ? „templateParameters”: parameters-common-map-v2 ; Zusätzliche Parameter der Vorlage.
}

parameters-common-map-v2 = (
  * tstr => any
)

parameters-map-v2 = {
  ? „redacted”: $jumbf-uri-type, ; Eine JUMBF-URI zur redigierten Assertion, erforderlich, wenn die Aktion
„c2pa.redacted” ist.
  ? „Zutaten“: [1* $hashed-uri-map], ; Eine Liste von gehashten JUMBF-URIs zu den Zutaten (v2 oder v3), auf
die diese Aktion wirkt
  ? „sourceLanguage“: tstr .size (1..max-tstr-length), ; BCP-47-Code der Ausgangssprache einer
`c2pa.translated`-Aktion
  ? „targetLanguage”: tstr .size (1..max-tstr-length), ; BCP-47-Code der Zielsprache einer `c2pa.translated`-
Aktion
  ? „multipleInstances”: bool, ; Wurde diese Aktion mehrfach ausgeführt? parameters-
  common-map-v2, ; Alles aus den gemeinsamen Parametern
}
```

Standardaktionen, die speziell für Schriftarten-Assets gelten, sind beschrieben in:

CDDL für Schriftartenaktionen

```
; Zuordnungen, Bereiche und Parameter für fontspezifische Aktionen.

; Mehrere Schriftartenaktionen funktionieren in Bezug auf Bereiche von Unicode-
Werten. font-unicode-range-map = {
  „start”: uint, ; Inklusive Start „stop”:
  uint, ; Inklusive Ende
}
```

```
; Schriftartparameter, der von font.subset, font.charactersAdded,
; font.charactersDeleted und font.charactersModified verwendet
wird. font-parameter-unicode-ranges-map = {
  „ranges”: [1* font-unicode-range-map] ; Array von Unicode-Bereichen
}

; Bereiche für Font-Instanziierungsparameter
font-weight-range = 1..1000 ; Gültige Gewichte oder Stärken für die Schriftart. 400 ist normal. font-
width-range = 0.0..1000.0 ; Prozentsatz des Normalwerts von 0 % bis 1000 %. 100 % ist die normale
Breite.
font-slant-range = -90,0..90,0 ; Neigungswinkel, wobei 0 Grad keine Neigung bedeutet.

; Schriftparameter, die beim Erstellen einer Instanz einer Schriftart aus einer variablen Schriftart verwendet
werden.
; Die verschiedenen „Variationsachsen” für die Schriftarten werden hier detailliert beschrieben. Die Tags
; für die verschiedenen Achsen stehen in Klammern in den Kommentaren zu den einzelnen
; Parameter angegeben.
font-parameter-created-from-variable-font-map = {
  ? „weight”: font-weight-range, ; Gewicht (wght) oder Stärke der zu instanziierenden Schriftart.
  ? „width”: font-width-range, ; Breite (wdth) oder Schmalheit der Buchstabenformen der zu instanziierenden
Schriftart.
  ? „italic”: bool, ; Ruft die kursive (ital) Version der Schriftart ab.
  ? „slant”: font-slant-range, ; Der Neigungswinkel (slnt) der Schriftart.
  ? „optical-size”: int / float, ; Die optische Größe (opsz) der Schriftart, in der Regel soll sie der gewünschten
Schriftgröße entsprechen.
  * tstr => beliebig ; Name und Typ der benutzerdefinierten Achsen.
}
```

Beispiel 11. Beispiel für eine v2-Aktion

Ein Beispiel für eine v2-Aktion in CBOR-Diagnoseschrift (RFC 8949, Abschnitt 8) ist unten aufgeführt:

```
{
  "actions": [
    {
      „action”: „c2pa.filtered”,

      „parameters”: {

      },
      „softwareAgent“: {
           „name“: „Joe's Photo Editor“, „version“: „2.0“,
           „operating_system“: „Windows 10“
      }
    },
    {
      „action“: „c2pa.cropped“,

    }
  ],
  „metadata“: {

    „reviewRatings”: [
      {
        „value”: 1,
        „Erklärung“: „Inhaltsbindungen wurden nicht validiert“
      }
    ]
  }
}
```

18.15. Zutat

18.15.1. Beschreibung

Wenn Assets zusammengesetzt werden, beispielsweise wenn ein Bild in eine Ebene in einem Bildbearbeitungsprogramm oder ein Audioclip in ein Video in einem Videobearbeitungsprogramm eingefügt wird, ist es wichtig, dass Informationen zu etwaigen Ansprüchen aus dem eingefügten Asset in das neue Asset aufgenommen werden, damit die gesamte Historie des neuen zusammengesetzten Assets nachvollziehbar ist. Dies gilt auch, wenn ein vorhandenes Asset zur Erstellung eines abgeleiteten Assets oder einer Asset-Wiedergabe verwendet wird.

Eine weitere häufige Verwendung für eine Zutat ist die Beschreibung einiger Assets oder Daten, die als Eingabe für einen Prozess verwendet wurden, wie beispielsweise die mit einem KI-/ML-Modell verbundenen Trainings- oder Inferenzanfragen.

Es gibt drei Versionen der Zutatenangabe: die ursprüngliche Version 1 (mit der Bezeichnung `c2pa.ingredient`), die verbesserte Version 2 (mit der Bezeichnung `c2pa.ingredient.v2`) und die weiter verfeinerte Version 3 (mit der Bezeichnung `c2pa.ingredient.v3`), die sich mit der Frage der Validierung von Zutaten nach der Redaktion befasst.

HINWEIS Da es höchstwahrscheinlich mehr als eine Inhaltsstoffangabe geben wird, ist die Verwendung der monotonischen steigend Index in der Etikett würde sein verwendet(z. B. `c2pa.ingredient.v3`, `c2pa.ingredient.v3 1`, `c2pa.ingredient.v3 2`).

18.15.2. Festlegung eindeutiger Identifikatoren

Wenn die hinzugefügte Zutat ein C2PA-Manifest enthält, muss ihre eindeutige Kennung aus dem Manifest-Etikett der JUMBF-Superbox entnommen werden, die das aktive C2PA-Manifest der Zutat enthält, und es ist nicht erforderlich, das optionale Feld „instanceID“ der Zutatenangabe anzugeben. Wenn der Anspruchgenerator das optionale Feld „instanceID“ der Inhaltsstoffangabe angibt, muss der Wert der eindeutigen Kennung gemäß Abschnitt 8.3, „Identifizierung von Nicht-C2PA-Assets“, festgelegt werden.

HINWEIS Ein Anspruchgenerator kann ein Feld „instanceID“ in der Inhaltsstofferklärung bereitstellen, auch wenn der Inhaltsstoff über ein C2PA-Manifest verfügt.

18.15.3. Beziehung

Beim Hinzufügen einer Zutat muss deren Beziehung zum aktuellen Vermögenswert beschrieben werden. Die möglichen Werte des Beziehungsfeldes und ihre Bedeutungen sind in Tabelle 10, „Beziehungen zwischen Inhaltsstoffen”, aufgeführt.

Tabelle 10. Beziehungen zwischen Inhaltsstoffen

Wert	Bedeutung
`parentOf`	Das aktuelle Asset ist ein abgeleitetes Asset oder eine Asset-Wiedergabe dieses Inhaltselements. Dieser Beziehungswert wird auch bei Aktualisierungsmanifesten verwendet.
`componentOf`	Das aktuelle Asset besteht aus mehreren Teilen, wobei diese Komponente einer davon ist.
`inputTo`	Diese Zutat wurde als Input für einen Berechnungsprozess verwendet, z. B. ein KI-/ML-Modell, der zur Erstellung oder Änderung dieses Assets geführt hat.

Beim Hinzufügen einer Inhaltsstoffangabe muss ein Anspruchgenerator eine `c2pa.`actions-Angabe hinzufügen (siehe Abschnitt 18.14, „Aktionen"), sofern diese nicht bereits im aktiven Manifest vorhanden ist. Je nach Art des Inhaltsstoffs muss einer der folgenden neuen Einträge zum Aktionsarray einer `c2pa.`actions-Angabe hinzugefügt werden.

- Beim Hinzufügen einer Zutat mit einer `parentOf`-Beziehung muss eine `c2pa.`opened-Aktion zum `Aktionsarray` hinzugefügt.
- Beim Hinzufügen einer Zutat mit einer „`componentOf`"-Beziehung muss eine „`c2pa.`placed"-Aktion zum `actions`-Array hinzugefügt.

Diese Anforderung gilt nur für Standard-Manifeste, da das Aufzeichnen von Aktionen nur in diesem Manifesttyp unterstützt wird.

18.15.4. Titel

Wenn vorhanden, muss der Wert von `dc:title` ein für Menschen lesbarer Name für die Zutat sein, der entweder aus dem XMP der Ressource oder dem Namen der Ressource in einem lokalen oder entfernten (z. B. cloudbasierten) Dateisystem stammen kann. Wenn die Zutat keinen bestimmten Namen hat, kann stattdessen eine Beschreibung der Zutat verwendet werden.

18.15.5. Format

Wenn vorhanden, muss der Wert von `dc:format` dem IANA-Medientyp für die Komponente entsprechen. Es wird empfohlen, dass ein Claim-Generator dieses Feld bereitstellt und dass es einen gültigen Wert enthält. Bei der Beschreibung einer Komponente mit mehreren Dateien, wie z. B. dem Datensatz eines KI-/ML-Modells, muss das Feld `dc:format` auf `multipart/mixed` gesetzt werden.

18.15.6. Schema und Beispiel

Die CDDL-Definition für diesen Typ lautet:

```
; Aussage, die eine in der Ressource verwendete Zutat beschreibt
ingredient-map = {
  „dc:title": tstr, ; Name der Zutat
  „dc:format": Formatzeichenfolge, ; Medientyp der Zutat
  ? „documentID": tstr, ; Wert von „xmpMM:DocumentID" der Zutat „instanceID": tstr, ; eindeutige
  Kennung, z. B. der Wert von
`xmpMM:InstanceID`
```

```
  „relationship”: $relation-choice, ; Beziehung dieses Inhaltselements zu dem Asset, zu dem es gehört.
  ? „c2pa_manifest”: $hashed-uri-map, ; hashed_uri-Verweis auf das C2PA-Manifest der Zutat
  ? „thumbnail”: $hashed-uri-map, ; hashed_uri-Verweis auf eine Zutaten-Miniaturansicht
  ? „validationStatus”: [1* $status-map] ; Validierungsstatus der Zutat
  ? „metadata”: $assertion-metadata-map ; zusätzliche Informationen zur Assertion
}

; Version 2 (v2) der Inhaltsstoffaussage
; Assertion, die eine in der Ressource verwendete Zutat beschreibt
ingredient-map-v2 = {
  „dc:title”: tstr, ; Name der Zutat
  „dc:format“: format-string, ; Medientyp der Zutat
  „relationship“: $relation-choice, ; Beziehung dieser Zutat zu dem Asset, dessen Bestandteil sie ist.
  ? „documentID”: tstr, ; Wert des Bestandteils „xmpMM:DocumentID”
  ? „instanceID”: tstr, ; eindeutige Kennung, z. B. der Wert von „xmpMM:InstanceID” der Zutat
`xmpMM:InstanceID`
  ? „data” : $hashed-uri-map / $hashed-ext-uri-map, ; hashed_uri-Verweis auf eine eingebettete Datenaussage
oder ein hashed_ext_uri auf externe Daten
  ? „data_types”: [1* $asset-type-map], ; zusätzliche Informationen zum Datentyp für die V2-Struktur der Zutat.
  ? „c2pa_manifest”: $hashed-uri-map, ; hashed_uri-Verweis auf das C2PA-Manifest der Zutat
  ? „thumbnail”: $hashed-uri-map, ; hashed_uri-Verweis auf eine Miniaturansicht in einer eingebetteten Datenaussage
  ? „validationStatus”: [1* $status-map] ; Validierungsstatus der Zutat
  ? „description”: tstr .size (1..max-tstr-length) ; Zusätzliche Beschreibung des Inhaltsstoffs
  ? „informational_URI”: tstr .size (1..max-tstr-length) ; URI zu einer Informationsseite über die Zutat oder
ihre Daten
  ? „metadata”: $assertion-metadata-map ; zusätzliche Informationen zur Assertion
}

; Version 3 (v3) der Inhaltsstoffangabe
; Angabe, die einen in der Anlage verwendeten Inhaltsstoff beschreibt
ingredient-map-v3 = {
  ? „dc:title”: tstr, ; Name der Zutat
  ? „dc:format”: format-string, ; Medientyp der Zutat
  „relationship”: $relation-choice, ; Beziehung dieser Zutat zu dem Asset, dessen Zutat sie ist.
  ? „validationResults”: $validation-results-map, ; Ergebnisse des Claim-Generators, der eine vollständige
Validierung des Inhalts-Assets durchführt
  ? „instanceID”: tstr, ; eindeutige Kennung, z. B. der Wert des Inhaltselements
`xmpMM:InstanceID`
  ? „data“: $hashed-uri-map / $hashed-ext-uri-map, ; hashed_uri-Verweis auf eine eingebettete Datenaussage
oder ein hashed_ext_uri auf externe Daten
  ? „dataTypes”: [1* $asset-type-map], ; zusätzliche Informationen zum Datentyp für die V3-Struktur der Zutat
  ? „activeManifest”: $hashed-uri-map, ; hashed_uri zum Feld, das dem aktiven Manifest der Zutat entspricht
  ? „claimSignature”: $hashed-uri-map, ; hashed_uri zum Feld „Claim Signature” im C2PA-Manifest der Zutat
  ? „thumbnail”: $hashed-uri-map, ; hashed_uri-Verweis auf eine Miniaturansicht in einer eingebetteten Datenaussage
  ? „description”: tstr .size (1..max-tstr-length), ; Zusätzliche Beschreibung der Zutat
  ? „informationalURI”: tstr .size (1..max-tstr-length), ; URI zu einer Informationsseite über die Zutat oder
ihre Daten
  ? „softBindingsMatched”: bool, ; Angabe, ob Soft-Bindings übereinstimmen
  ? „softBindingAlgorithmsMatched”: [1* tstr] ; Array der Algorithmusnamen, die für die
das aktive Manifest zu ermitteln
```

```
  ? „metadata”: $assertion-metadata-map ; zusätzliche Informationen zur Assertion
}

format-string = tstr .regexp "^\\w+\/[-+.\\w]+$"

; Auswahlmöglichkeiten, die beschreiben, in welcher Beziehung die Zutat zu dem Vermögenswert steht
$relation-choice /= „parentOf”
$relation-choice /= „componentOf”
$relation-choice /= „inputTo”
```

Ein Beispiel in CBOR-Diagnoseschrift (RFC 8949, Abschnitt 8):

```
{
  "dc:title": "image 1.jpg", "metadata": {

    „reviewRatings”: [
      {
        „value”: 5,
        „explanation”: „Inhaltsbindungen validiert”
      }
    ]
  }
  „dc:format“: „image/jpeg“, „thumbnail“: {
       „URL“: „self#jumbf=c2pa.assertions/c2pa.thumbnail.ingredient“, „Hash“:
       b64'UjRAYWiAq4lfCRDmksWAlDJN/XtHHFFwMWymsZsm3j8='
  },
  „Beziehung“: „parentOf“, „activeManifest“:
  {
       "url": "self#jumbf=/c2pa/urn:c2pa:5E7B01FC-4932-4BAB-AB32-D4F12A8AA322", "hash":
       b64'1kjJTO108b71cL95UxgfHD3eDgk9VrCedW8n3fYTRMk='
  },
  „claimSignature”: {


  },
  „validierungsergebnisse“: { „aktives
    Manifest“: {
      "success": [
      {
        „code”: „claimSignature.validated”,

      },{
        „code”: „signingCredential.trusted”,

      },{
        „code”: „timeStamp.validated”,

      },{
        „code”: „timeStamp.trusted”,

      },{
        „code”: „assertion.hashedURI.match”,
```

```
D4F12A8AA322/c2pa.assertions/c2pa.ingredient.v3"
        }
        ],
        „informational”: [{
           "code": "signingCredential.ocsp.skipped",

        }],
        „Fehler”: []
     },
     „ingredientDeltas“: [{
        „ingredientAssertionURI”: „self#jumbf=/c2pa/urn:c2pa:5E7B01FC-4932-4BAB-AB32-
D4F12A8AA322/c2pa.assertions/c2pa.ingredient.v3“,
        „validationDeltas”: { „success”:
           [],
           „informational”: [],
           „failure”: [{
              "code": "assertion.hashedURI.mismatch",

        }]
        }
     },{
        „ingredientAssertionURI”: „self#jumbf=/c2pa/urn:c2pa:F095F30E-6CD5-4BF7-8C44-
CE8420CA9FB7/c2pa.assertions/c2pa.ingredient.v3“,
        „validierungsdeltas“: {
           „erfolgreich“: [],
           „informational”: [],
           „Fehler”: [{
              „code”: „signingCredential.untrusted”,

        }]
        }
     }
     ]
   }
}
```

18.15.7. Beschreibungsfeld

Eine Zutat kann im Beschreibungsfeld eine Freitextbeschreibung enthalten, die angibt, um was es sich bei der Zutat handelt oder wofür sie verwendet wird. Dies ist nützlich in Situationen, in denen weder der Titel noch das Format ausreichend sind.

18.15.8. Zutaten-Daten

18.15.8.1. Standardverwendung

In bestimmten Anwendungsfällen, wie z. B. generativer KI, kann es wichtig sein, über Zutaten zu verfügen, bei denen die Daten der Zutat bereitgestellt werden – entweder eingebettet in das C2PA-Manifest oder über eine URL, die auf die Daten verweist. Dies wird durch das Datenfeld in der Zutat erreicht, das eine `Hash-URI` verwendet, um auf eine eingebettete Datenaussage zu verweisen, oder eine `Hash-Ext-URI,` um auf eine externe Referenz zu verweisen.

HINWEIS Frühere Versionen dieser Spezifikation erlaubten es, dass die `Hash-URI` auf eine Datenbox verweist.

Die Verwendung einer eingebetteten Datenaussage bedeutet, dass ihr Inhalt in dieses C2PA-Manifest und in jedes zukünftige C2PA-Manifest (sofern nicht redigiert) eingebettet wird, das diese Ressource als Bestandteil enthält. Bei der Entscheidung, ob Daten eingebettet werden sollen, sollten Claim-Generatoren die Größe dieses Feldes berücksichtigen.

Beispiel 12. Beispiel für Bestandteile mit Daten

Ein Beispiel für einige Inhaltsstoffe mit Daten in CBOR-Diagnoseschrift (RFC 8949, Abschnitt 8):

```
// Datenfeld der Eingabeaufforderung //
{
  „dc:format”: „text/plain”,
  „data” : 'Pirate mit Vogel auf der Schulter' „dataTypes”:
  [{
    "type": "c2pa.types.generator.prompt",
  }]
}

// Zutat (Eingabeaufforderung) //
{
  „dc:title“: „prompt“, „dc:format“:
  „text/plain“, „relationship“:
  „inputTo“, „data“: {
    "url": "self#jumbf=c2pa.assertions/c2pa.embedded-data", "alg" :
    "sha256",
    „hash“: b64'...',
  }
}


// Zutat (Modell) //
{
  „dc:title”: „Modell”,
  „dc:format“: „application/octet-stream“, „dataTypes“: [
    {
      "type": "c2pa.types.generator",
    },
    {
      „type”: „c2pa.types.model.tensorflow”, „version”:
      „1.0.0”,
    },
    {
      „type”: „c2pa.types.tensorflow.hubmodule”, „version”:
      „1.0.0”,
    }
  ],
  „Beziehung“: „inputTo“, „Daten“: {
    „url“: „https://tfhub.dev/deepmind/bigbigan-resnet50/1?tf-hub-format=compressed“, „alg“: „sha256“,
    „hash“: b64'...',
  },
  „description”: „Unüberwachtes BigBiGAN-Bildgenerierungs- und Repräsentationslernmodell, trainiert
auf ImageNet mit einer kleineren (ResNet-50) Encoder-Architektur.”,
  „informationalURI“: „https://tfhub.dev/deepmind/bigbigan-resnet50/1“,
}
```

Es gibt auch Anwendungsfälle, in denen es wichtig ist, Informationen über die Daten der Zutat zu identifizieren, es jedoch weder möglich ist, diese einzubetten noch eine gültige URL anzugeben – beispielsweise bei der Beschreibung der Verwendung eines privaten/internen KI-Modells. In diesen Fällen kann ein Asset-Typ als Wert des Feldes `„data_types“` angegeben werden, um mehr Klarheit über das Format und die Beschreibung dieser Daten zu schaffen.

Beispiel 13. Beispiel für Zutaten mit data_types

Ein Beispiel für eine Zutat ohne Hash-URI in CBOR-Diagnoseschrift (RFC 8949, Abschnitt 8):

```
// Zutat (privates Modell) //
{
     "dc:title": "model",
     "dc:format": "application/octet-stream", "relationship":
     "inputTo",
     „dataTypes”: [
          {
               „type”: „c2pa.types.generator”,
          },
          {
               „type”: „c2pa.types.model.tensorflow”, „version”:
               „1.5.0”,
          }
     ],
     „description”: „Joes privates generatives KI-Modell”, „informationalURI”:
     „https://www.example.com/joes-model-info.html”
}
```

18.15.8.2. Mehrere Dateien als Bestandteile

In einigen Fällen kann eine Zutat als eine Reihe mehrerer Dateien dargestellt werden, beispielsweise als Trainingsdatensatz für ein KI-/ML-Modell. In diesen Fällen wird empfohlen, das C2PA-Manifest in die Zutatenangabe aufzunehmen und das C2PA-Manifest für den vollständigen Datensatz mit einer Asset-Referenzangabe zu versehen, die verweist, wo diese Dateien zu finden sind.

HINWEIS Diese Methode eignet sich besonders für die Arbeit mit einer Sammlung von Assets, bei denen nicht alle Dateien in derselben Hierarchie enthalten sind.

18.15.9. Informations-URI

Wenn es erforderlich ist, eine URL zu einer Webseite mit Informationen über die Komponente anzugeben, z. B. detaillierte Informationen über ein KI-/ML-Modell, sollte diese als Wert des Feldes `„informationalURI”` der Komponentenaussage angegeben werden.

HINWEIS Die `informationalURI` ist kein authentifizierter Link zum Inhalt der Zutat selbst, sondern etwas, das für einen menschlichen Benutzer von allgemeinem Interesse ist.

HINWEIS Ältere (und veraltete) Versionen von der Zutat Aussage mit dem Namen dieses Feld `informational_URI`.

18.15.10. Miniaturansichten

Beim Hinzufügen einer Zutat kann es sinnvoll sein, auch eine Miniaturansicht der Zutat hinzuzufügen, um den Zustand der Zutat zum Zeitpunkt des Imports zu dokumentieren. Zu diesem Zweck wird eine Miniaturansicht als Miniaturansicht-Assertion hinzugefügt und hier über eine Hash-URI-Referenz referenziert.

Manifest-Konsumenten sollten auch den in früheren Versionen dieser Spezifikation empfohlenen Datenbox-Ansatz unterstützen.

18.15.11. Bestehende Manifeste

18.15.11.1. Allgemeines

Wenn für die Komponente bereits ein C2PA-Manifest-Speicher vorhanden ist, müssen alle C2PA-Manifeste im C2PA-Manifest-Speicher der Komponente, die validiert wurden und noch nicht im C2PA-Manifest-Speicher des Assets vorhanden sind, vom Claim-Generator in den C2PA-Manifest-Speicher des Assets kopiert werden, außer wie in Abschnitt 18.15.12 beschrieben, „Kopieren vorhandener Manifeste" oder wenn dies nicht gewünscht ist (z. B. durch Benutzereingabe oder Konfiguration).

Der Anspruchgenerator sollte außerdem alle zusätzlichen C2PA-Manifeste, die nicht validiert wurden, sowie alle zusätzlichen JUMBF-Boxen und Superboxen, die im C2PA-Manifest-Speicher erscheinen und nicht als C2PA-Manifeste erkannt werden, in den C2PA-Manifest-Speicher des Assets kopieren.

HINWEIS Das Kopieren dieser zusätzlichen Elemente unterstützt die Kompatibilität mit benutzerdefinierten Assertions und zukünftigen Konstrukte, die möglicherweise auf Elemente des C2PA-Manifest-Speichers in einer Weise verweisen, die der Anspruchgenerator nicht erkennt.

18.15.12. Kopieren vorhandener Manifeste

18.15.12.1. Feststellen des Bedarfs

Um festzustellen, ob ein vorhandenes Manifest aus dem C2PA-Manifest-Speicher der Zutat in den C2PA-Manifest-Speicher des Assets kopiert werden muss, muss der Anspruchgenerator:

1. die Zutat gemäß dem in Abschnitt 18.15.12.4 „Validierung von Zutaten" beschriebenen Verfahren validieren. Im Falle einer fehlgeschlagenen Validierung kann der Anspruchgenerator die restlichen Schritte überspringen, wenn er dazu aufgefordert wird (z. B. durch Benutzereingabe oder Konfiguration).
2. Für jedes Manifest im C2PA-Manifest-Speicher der Zutat die URN-Kennung mit den URN-Kennungen jedes C2PA-Manifests vergleichen, das bereits im C2PA-Manifest-Speicher des Assets vorhanden ist.
    a. Wenn eine Übereinstimmung gefunden wird, berechnen und vergleichen Sie den Hashwert des Manifestfelds aus dem C2PA-Manifest-Speicher der Zutat mit dem Hashwert des übereinstimmenden Manifestfelds aus dem C2PA-Manifest-Speicher des Assets.
        i. Wenn die Hash-Werte übereinstimmen, darf der Anspruchgenerator das Manifest nicht aus dem C2PA-Manifest-Speicher der Komponente in den C2PA-Manifest-Speicher des Assets kopieren.
        ii. Wenn die Hash-Werte nicht übereinstimmen:

A. Der Anspruchgenerator überprüft, ob Aussagen aus einem der beiden Manifeste redigiert wurden (optional unter Verwendung der Liste der Redaktionen, die im Prozess „Durchführung einer expliziten Validierung" zusammengestellt wurde).

I. Wenn der Validierer feststellen kann, dass die Hashes nur aufgrund der Redigierung voneinander abweichen, dann:

1. Wenn alle Redaktionen auf das bereits im C2PA-Manifest-Speicher des Assets vorhandene Manifest angewendet wurden, darf der Anspruchgenerator das Manifest nicht aus dem C2PA-Manifest-Speicher der Komponente in den C2PA-Manifest-Speicher des Assets kopieren.

2. Wenn alle Redaktionen auf das Manifest aus dem Manifest-Speicher der Komponente angewendet wurden, ersetzt der Anspruchgenerator das Manifest im C2PA-Manifest-Speicher des Assets durch das Manifest aus dem C2PA-Manifest-Speicher der Komponente.

3. Wenn unterschiedliche Redaktionen sowohl auf das C2PA-Manifest aus dem C2PA-Manifest-Speicher der Zutat als auch auf das C2PA-Manifest aus dem C2PA-Manifest-Speicher des Vermögenswerts angewendet wurden, muss der Anspruchgenerator so viele Aussagen wie nötig aus dem bestehenden Manifest im C2PA-Manifest-Speicher des Vermögenswerts redigieren, dass eine Vereinigung der beiden Redaktionssätze entsteht.

II. In allen anderen Fällen kopiert der Anspruchgenerator das Manifest aus dem C2PA-Manifest-Speicher der Zutat, kennzeichnet es gemäß dem in „Eindeutige Identifikatoren" beschriebenen Verfahren mit einer aktualisierten URN neu und fügt die neu gekennzeichnete Version in den C2PA-Manifest-Speicher des Vermögenswerts ein.

Die Entscheidung, ob die Hashes nur aufgrund der Schwärzung voneinander abweichen, bleibt dem Validator überlassen.

HINWEIS

18.15.12.2. Beispiele

BEISPIEL: Betrachten wir einen Anspruchgenerator D, der Inhaltsstoffe importiert. Er beginnt mit dem Import des Inhaltsstoffs B, der selbst einen Inhaltsstoff-Manifest A enthält. Nach der Validierung beider Manifeste kopiert der Anspruchgenerator D die Manifeste B und A in den C2PA-Manifest-Speicher von Asset D. Anschließend importiert er die Zutat C, die ebenfalls eine redigierte Version des Manifests A enthält. Nach der Validierung des Manifests C und des redigierten Manifests A vergleicht er die Hash-Werte beider Versionen des Manifests A. Da er weiß, dass die Version des Manifests A in der Zutat C redigiert wurde, überschreibt der Anspruchgenerator D die bereits im C2PA-Manifest-Speicher von Asset D vorhandene Version des Manifests A mit der redigierten Version des Manifests A aus der Zutat C.

BEISPIEL: Betrachten Sie das gleiche Szenario wie oben, außer dass die Version von Manifest A in Bestandteil C die Validierung nicht bestanden hat, weil eine ihrer Aussagen einen Hash-Vergleich nicht bestanden hat. In dieser Situation kopiert der Anspruchgenerator D Manifest A aus Bestandteil C, kennzeichnet es mit einer neuen URN und legt die neu gekennzeichnete Kopie im C2PA-Manifest-Speicher von Asset D ab.

HINWEIS

Ein C2PA-Manifest-Speicher kann JUMBF-Boxen oder Superboxen enthalten, die keine C2PA-Manifeste sind. Diese müssen im Rahmen dieses Prozesses nicht kopiert werden.

18.15.12.3. Hinzufügen von Manifestreferenzen zur Inhaltsstoffaussage

Wenn das aktive Manifest der Zutat in den C2PA-Manifest-Speicher des Assets kopiert wurde, muss eine URI-Referenz auf die aktive C2PA-Manifest-Box der Zutat als Wert des Feldes „activeManifest" in der Zutatenaussage gespeichert werden, und eine zusätzliche URI-Referenz auf die C2PA-Claim-Signatur-Box des aktiven Manifests muss als Wert von „claimSignature" gespeichert werden.

Für ein C2PA-Manifest, das im C2PA-Manifest-Speicher vorhanden ist, müssen hashed_uri`s als Werte für die Felder `activeManifest und claimSignature der Inhaltsstoffangabe verwendet werden.

HINWEIS

Die Angabe beider Werte ermöglicht eine effiziente Validierung der Inhaltsstoffe und unterstützt auch die Validierung, wenn eine der Inhaltsstoffangaben redigiert wurde.

18.15.12.4. Zutatenvalidierung

18.15.12.4.1. Allgemeines

Wenn die Inhaltsstoffangabe auf ein C2PA-Manifest verweist, fungiert der Anspruchgenerator zusätzlich als Validierer und führt die Validierung des Inhaltsstoffs gemäß den Validierungsschritten durch. Das Ergebnis dieser Validierung – alle Erfolgscodes, Informationscodes und Fehlercodes – wird zur Befüllung des Feldes „validationResults" oder „validationStatus" der Inhaltsstoffangabe verwendet, wie unten beschrieben. Dieses Feld muss vorhanden sein, damit es für zukünftige Validierungen verwendet werden kann.

HINWEIS

Das Vorhandensein eines validationStatus (Zutat v2) oder validationResults (Zutat v3) mit einem Fehlerstatus wird als ausdrückliche Erklärung des Anspruchgenerators angesehen, dass ein Akteur Validierungsfehler im C2PA-Anspruch der Zutat selbst anerkannt hat und sich dafür entschieden hat, mit
die Zutat zu integrieren.

Wie in Abschnitt 15.3, „Anzeigen von Manifestinformationen", beschrieben, ist es wünschenswert, dass ein Anspruchgenerator deutlich sichtbare Warnungen ausgibt, damit ein Akteur, der eine Ressource mit einer fehlerhaften Herkunftshistorie nutzt, eine fundierte Entscheidung darüber treffen kann, wie er vorgehen möchte.

18.15.12.4.2. V2-Zutatenangaben (VERALTET)

In einer V2-Zutatenangabe ohne c2pa_manifest-Feld ist das validationStatus-Feld optional, kann jedoch, falls vorhanden, ein leeres Array enthalten.

In einer v2-Zutatenangabe mit dem Feld „c2pa_manifest" besteht jedes Objekt im Array „validationStatus" aus einem Feld „code", dessen Wert den Validierungsstatus eines bestimmten Teils des Manifests beschreibt, sowie einem optionalen Feld „success", dessen boolescher Wert angibt, ob der Code einen Erfolg (true) oder einen Fehler (false) widerspiegelt. Eine optionale Beschreibung des Validierungsstatus kann im Feld „explanation" vorhanden sein, wenn eine zusätzliche, für Menschen lesbare Erklärung erforderlich ist. Darüber hinaus verfügt jedes status-map-Objekt über ein url-Feld, das im Falle eines Fehlers eine JUMBF-URI-Referenz zu dem spezifischen Element im Manifest enthalten sollte, auf das sich der Status bezieht. Je nach Code verweist die URL auf einen C2PA-Claim, eine C2PA-Claim-Signatur oder eine spezifische C2PA-Assertion. Statuscodes sind in Abschnitt 15.2.2, „Standard-Statuscodes", definiert.

Benutzerdefinierte Statuscodes sind zulässig, wenn ein Anspruchgenerator prozessspezifische Statusinformationen aufzeichnen muss. Der Code muss derselben Syntax entsprechen wie entitätsspezifische Namespaces (z. B. `com.litware.malformedFrobber`), und das validationStatus-Objekt muss einen booleschen Wert `für „Erfolg“` enthalten.

18.15.12.4.3. V3-Inhaltsstoff-Assertions

In einer v3-Zutatenangabe ohne Feld `„activeManifest”` darf das Feld `„validationResults”` nicht vorhanden sein. In einer v3-Zutatenangabe mit einem Feld `„activeManifest”` muss das Feld `„validationResults”` ein `validation-results-map`-Objekt enthalten, das wiederum Folgendes enthält:

1. In `activeManifest` die vollständigen Validierungsergebnisse für das aktive Manifest der Zutat.
2. In `ingredientDeltas` die Delta-Validierungsergebnisse für jede Inhaltsstoffangabe, die ein Feld `„activeManifest“` enthält, in jedem Manifest im C2PA-Manifest-Speicher des Inhaltsstoffs. Die Delta-Validierungsergebnisse für eine Inhaltsstoffangabe müssen Folgendes enthalten:
   a. In `ingredientAssertionURI` die URI der Inhaltsstoffangabe.
   b. In `validationDeltas` die Validierungsergebnisse für das Manifest, auf das sich die Inhaltsstoffangabe bezieht, wobei alle Statuswerte im Feld `activeManifest` des Feldes `validationResults` in der Inhaltsstoffangabe (oder bei Inhaltsstoffangaben der Versionen v1 oder v2 im Feld `validationStatus`) weggelassen werden. Bei diesem Statuswertvergleich sind der Statustyp (Erfolg, Information oder Fehler), `der Code` und `die URL` zu berücksichtigen, während andere Felder ignoriert werden.

BEISPIEL: Betrachten wir ein Manifest `E` mit mehreren Inhaltsstoffen und einer komplexen Herkunft. Der Anspruchgenerator `E` fügt Manifest `C` und Manifest `D` über Inhaltsstoffaussagen als Inhaltsstoffe hinzu. Manifest C selbst hat Manifest A und Manifest B über Inhaltsstoffangaben hinzugefügt. Manifest D hat ebenfalls Manifest A über eine Inhaltsstoffangabe hinzugefügt. Beim Hinzufügen von Manifest C erstellt der Anspruchgenerator E eine Inhaltsstoffangabe mit einem validationResults-Objekt, das die Validierungsergebnisse für das aktive Manifest von C in `activeManifest` und die Delta-Validierungsergebnisse für die Manifeste A und B in `ingredientDeltas` speichert. Das ingredientDeltas-Array enthält zwei Elemente: eines für die Delta-Ergebnisse im Vergleich zum `active`Manifest-Objekt im validationResults-Objekt in der Zutatenaussage von Manifest B in Manifest C (mit einem Hash-URI-Link zu dieser Zutatenaussage in Manifest C) und ein weiteres Element mit denselben Attributen, jedoch für die Zutatenaussage von Manifest A in Manifest C. Ebenso erstellt der Anspruchgenerator E beim Hinzufügen `von` Manifest D erstellt der Anspruchgenerator E eine Inhaltsstoffaussage, die sowohl `die Validierungsergebnisse` für `das aktive Manifest` von D als auch `die ingredientDeltas` mit einem einzigen Array-Element speichert, das die Delta-Validierungsergebnisse enthält, die mit dem Objekt `„activeManifest”` im Objekt `„validationResults”` in der Inhaltsstoffaussage von Manifest D zu Manifest A verglichen wurden.

HINWEIS Dies ist zwar ein bewusst konstruiertes Beispiel, es soll jedoch verdeutlichen, wie die Datenstruktur `„validationResults“` verwendet werden soll.

Jedes Validierungsergebnis (wie unter Verwendung einer `Status-Codes-Map` beschrieben) besteht aus einem Array von `Erfolgs-`, `Informations-` und Fehlercodes. Jeder Code wird als Status-Map-Objekt dargestellt, das ein Code-Feld mit dem Statuscode enthalten muss. Darüber hinaus kann es ein URL-Feld mit einer JUMBF-URI-Referenz zu dem spezifischen

Element im Manifest, auf das sich der Status bezieht, und ein optionales Erklärungsfeld mit einer für Menschen lesbaren Erklärung des Status. Statuscodes sind in Abschnitt 15.2.2, „Standard-Statuscodes“, definiert.

Benutzerdefinierte Statuscodes sind zulässig, wenn ein Anspruchgenerator prozessspezifische Statusinformationen aufzeichnen muss. Der Code muss derselben Syntax entsprechen wie entitätsspezifische Namespaces (z. B. `com.litware.malformedFrobber`).

18.15.13. Metadaten zu Inhaltsstoffen

Wie in den Metadaten der Assertion beschrieben, kann das Metadatenfeld der Assertion zu einer Zutat Metadaten über die Zutat enthalten, z. B. das Datum und die Uhrzeit ihrer Erstellung oder andere Daten, die Manifest-Verbrauchern helfen können, fundierte Entscheidungen über die Herkunft oder Richtigkeit der Assertion-Daten zu treffen.

Eine häufige Verwendung für das Metadatenfeld ist, wenn nur ein Teil einer Zutat bei der Erstellung oder Bearbeitung eines Assets verwendet wird. In solchen Fällen sollte das Metadatenfeld ein regionOfInterest-Feld enthalten (wie in Abschnitt 18.3.6, „Region of Interest“ beschrieben), das die relevanten Teile der verwendeten Zutat beschreibt. Ein Beispiel hierfür finden Sie in Beispiel 14, „Beispiel für eine Zutat mit Metadaten, die Regionen enthalten“.

HINWEIS

Obwohl das Feld nur einen einzigen Bereich von Interesse enthält, kann das region-map-Objekt Folgendes angeben mehrere Regionen als Werte für sein Feld „`region`“ angeben. Dies ist nützlich, wenn mehrere Teile einer einzelnen Zutat beteiligt sind.

Beispiel 14. Beispiel für eine Zutat mit Metadaten, die Regionen enthalten

Ein Beispiel für eine Zutat, deren Metadaten einen Bereich von Interesse enthalten, in CBOR-Diagnoseschrift (RFC 8949, Abschnitt 8):

```
{
  „dc:title”: „someVideo.mp4”, „metadata”:
  {
    „regionOfInterest” : {
         „Beschreibung“: „10 Sekunden Audio“, „Region“: [
              {
                   „type“: „temporal“, „time“: {
                        „Typ“: „npt“,
                        „start“: „10“,
                        „end“: „20“
                   }
              },
              {
                   „Typ“: „identifiziert“, „Element“:
                   {
                        „Bezeichner“: „track_id“, „Wert“:
                        „3“
                   }
              }
         ]
    }
  }
  „dc:format“: „video/mp4“,
```

```
    „Beziehung“: „componentOf“,
    „activeManifest“: {
         "url": "self#jumbf=/c2pa/urn:c2pa:98782815-5116-4d78-93de-3f5d8b4f4615", "hash":
         b64'TEWww2UCIR/e8mmR0XvzkFVZYTJ59Q8Ip4nkYxrS/Ys='
    },
    „claimSignature” : {

         „hash”: b64'ICJkYzpmb3JtYXQiOiAiaW1hZ2UvanBlZyIsCiAgImR='
    },
    „validierungsergebnisse”: { ... }
}
```

18.15.14. Weiche Bindungen

Ein aktives Manifest kann ein C2PA-Manifest als Bestandteil (über eine parentOf-Beziehung) enthalten, das mithilfe einer Soft-Binding-Suche ermittelt wurde. Wenn der Claim Generator ein solches C2PA-Manifest enthält, muss er ein Feld `softBindingsMatched` mit dem Wert „true“ und ein Feld `softBindingAlgorithmsMatched` mit einer Reihe von Zeichenfolgen (mit den Namen der Soft-Binding-Algorithmen, die zur Ermittlung des C2PA-Manifests als Bestandteil verwendet wurden) enthalten. Die Algorithmusnamen müssen in der C2PA-Soft-Binding-Algorithmusliste aufgeführt sein, wie sie im Feld `„alg“` der Einträge in dieser Liste angegeben ist.

18.16. Metadaten

18.16.1. Beschreibung

In früheren Versionen dieser Spezifikation gab es einzelne Assertions für jeden Metadatenstandard (z. B. IPTC, EXIF). In dieser Version gibt es nun eine Kategorie von Assertions, die zur Darstellung von Metadaten in einer standardisierten Serialisierung verwendet werden sollen. Durch die Aufnahme der Metadaten in eine Assertion wird festgestellt, dass die Metadaten in dieser Assertion von Bedeutung sind, da sie ausdrücklich in das C2PA-Manifest aufgenommen und von einem bestimmten Unterzeichner signiert wurden, wodurch eine kryptografische Validierung und Zuordnung der Daten ermöglicht wird. Darüber hinaus ermöglicht die Verwendung einer gemeinsamen Serialisierung den Manifest-Nutzern eine einheitliche Verarbeitung.

HINWEIS Diese Aussagen können bestehende Standards oder private Spezifikationen darstellen.

18.16.2. Allgemeine Anforderungen

Eine Metadaten-Aussage muss eine Bezeichnung haben, die mit der Zeichenfolge `.metadata` endet und der entweder die Standardkennung `c2pa` oder eine andere Kennung vorangestellt ist, sofern diese der gleichen Syntax wie entitätsspezifische Namespaces entspricht. Beispielsweise wäre eine Aussage `com.litware.metadata` gültig.

Jede Metadaten-Aussage muss ein einzelnes JSON-Inhaltstypfeld enthalten, das die JSON-LD-Serialisierung eines oder mehrerer Metadatenwerte enthält. Die Eigenschaft `@context` innerhalb des JSON-LD-Objekts muss enthalten sein und wird verwendet, um Kontext/Namensräume für die angegebenen Metadatenstandards bereitzustellen. Das empfohlene Verfahren zum Erstellen dieses JSON-LD-Objekts besteht darin, zunächst eine XMP-Datenmodelldarstellung der Metadaten zu erstellen und diese dann gemäß der JSON-LD-Serialisierung von XMP in JSON-LD zu serialisieren. Das JSON-LD würde dann als JSON-Inhaltstyp-Box gespeichert werden.

18.16.3. Die `c2pa.metadata`-Aussage

Diese Spezifikation definiert eine Metadaten-Aussage mit der Bezeichnung „`c2pa.metadata`“, die zur Darstellung einer Teilmenge gängiger Metadatenschemata verwendet wird, die in jedem C2PA-Manifest verwendet werden können. Die Metadatenfelder, die in dieser Aussage enthalten sein können, sind in Anhang B, *„Implementierungsdetails für `c2pa.metadata`“*, dokumentiert.

HINWEIS Benutzerdefinierte Metadaten-Assertions können beliebige Werte aus beliebigen Schemata enthalten.

Beispiel 15. `c2pa.metadata`-Aussage für ein Bild

Ein Beispiel für eine `c2pa.metadata`-Aussage für ein Bild:

```
{
     „@context“ : {
          „exif”: „http://ns.adobe.com/exif/1.0/”, „exifEX”:
          „http://cipa.jp/exif/2.32/”, „tiff”:
          „http://ns.adobe.com/tiff/1.0/”,
          „Iptc4xmpExt“: „http://iptc.org/std/Iptc4xmpExt/2008-02-29/“, „photoshop“:
          „http://ns.adobe.com/photoshop/1.0/“
     },
     „photoshop:DateCreated”: „31. August 2022”, „Iptc4xmpExt:DigitalSourceType”:
„http://cv.iptc.org/newscodes/digitalsourcetype/digitalCapture“,
     „exif:GPSVersionID“: „2.2.0.0“,
     „exif:GPSLatitude“: „39,21.102N“,
     „exif:GPSLongitude“: „74,26.5737W“,
     „exif:GPSAltitudeRef“: 0, „exif:GPSAltitude“:
     „100963/29890“, "exif:GPSTimeStamp":
     "18:22:57",
     „exif:GPSDateStamp”: „2019:09:22”,
     „exif:GPSSpeedRef”: „K”, „exif:GPSSpeed”:
     „4009/161323”, „exif:GPSImgDirectionRef”: „T”,
     „exif:GPSImgDirection”: „296140/911”,
     „exif:GPSDestBearingRef”: „T”,
     „exif:GPSDestBearing”: „296140/911”,
     „exif:GPSHPositioningError“: „13244/2207“,
     „exif:ExposureTime“: „1/100“, „exif:FNumber“: 4.0,
     „exif:ColorSpace”: 1,
     „exif:DigitalZoomRatio”: 2.0, „tiff:Make”:
     „CameraCompany”, „tiff:Model”: „Shooter S1”,
     „exifEX:LensMake”: „CameraCompany”,
     „exifEX:LensModel”: „17,0–35,0 mm”,
     „exifEX:LensSpecification”: { „@list”: [ 1,55, 4,2, 1,6, 2,4 ] }

}
```

Beispiel 16. `c2pa.metadata`-Aussage für eine PDF-Datei

Ein Beispiel für eine `c2pa.metadata`-Aussage für eine PDF-Datei:

```
{
```

```
      „@context“: {
         "dc" : "http://purl.org/dc/elements/1.1/", "xmp" :
         "http://ns.adobe.com/xap/1.0/", "pdf" :
         "http://ns.adobe.com/pdf/1.3/", "pdfx":
         "http://ns.adobe.com/pdfx/1.3/"
      },
      „dc:created“: „3. Februar 2015“, „dc:title“: [
           „Dies ist eine Testdatei“
      ],
      „xmp:CreatorTool“: „TeX“, „pdf:Producer“:
      „pdfTeX-1.40.14“, „pdf:Trapped“: „Unbekannt“,
      „pdfx:PTEX.Fullbanner“: „Dies ist pdfTeX, Version 3.1415926-2.5-1.40.14 (TeX Live 2013) kpathsea
Version 6.1.1“
}
```

18.16.4. Bearbeitung von `c2pa.metadata`

Obwohl der Redaktionsprozess so funktioniert, dass nur eine gesamte Aussage redigiert werden kann (siehe Abschnitt 6.8, „Redaktion von Aussagen“), ermöglicht die Verwendung eines Aktualisierungsmanifests eine teilweise Redaktion, indem das Original entfernt und dann die neuen, reduzierten Versionen in das Aktualisierungsmanifest aufgenommen werden. Diese neue Behauptung würde in einer Benutzererfahrung in Verbindung mit dem Unterzeichner des Aktualisierungsmanifests und nicht mit dem Unterzeichner des redigierten C2PA-Manifests dargestellt werden.

Beispielsweise könnte eine Metadaten-Aussage, die sowohl Standortdaten als auch Kamerainformationen enthält und bei der die Standortdaten redigiert werden müssen, durch ein Aktualisierungsmanifest mit einer neuen Metadaten-Aussage ersetzt werden, die nur die Kamerainformationen enthält.

18.17. Zeitstempel

18.17.1. Beschreibung

In einigen Provenienz-Workflows wird ein Standard- oder Aktualisierungsmanifest offline erstellt, wobei es nicht möglich ist, zum Zeitpunkt der Signatur einen vertrauenswürdigen Zeitstempel (gemäß RFC 3161) von einer TSA zu erhalten. In solchen Fällen laufen diese Signaturzertifikate jedoch nach einer bestimmten Zeit ab, was zu einem ungültigen C2PA-Manifest führt.

Um dieses Ablaufen zu verhindern, kann zu einem späteren Zeitpunkt (sofern das Zertifikat noch nicht abgelaufen ist) ein vertrauenswürdiger Zeitstempel hinzugefügt werden, der einen „Existenzbeweis“ für dieses C2PA-Manifest und (im Falle des aktiven Manifests) die damit verbundene Ressource liefert. Diese Zeitstempel-Assertion wird verwendet, um einen vertrauenswürdigen Zeitstempel für solche C2PA-Manifeste bereitzustellen.

18.17.2. Schema und Beispiel

Das Schema für diesen Typ wird durch die Zeitstempel-Zuordnungsregel in der folgenden CDDL-Definition definiert:

```
; Die Datenstrukturen, die zum Speichern eines Arrays von
; Manifest-URNs in Zeitstempel-„Blobs“ zu speichern
$time-stamp-map /= {
```

```
  * $$time-stamp-entry => bstr
}

time-stamp-entry /= tstr .regexp "^urn:c2pa:[\\da-zA-Z_-]+$"
```

Ein Beispiel in CBOR-Diagnoseschrift (RFC 8949, Abschnitt 8) ist unten aufgeführt:

```
{
  „urn:c2pa:d61c74e0-ce26-4439-b92d-690dcce6b58e” : h'...',
  „urn:c2pa:ab8c2751-8711-455a-9a8b-37143bfc92c2” : h'...'
}
```

18.17.3. Anforderungen

Eine Zeitstempel-Assertion muss die Bezeichnung `c2pa.time-stamp` haben, und es darf höchstens eine Zeitstempel-Assertion pro C2PA-Manifest vorhanden sein.

Die Zeitstempel-Assertion besteht aus einer CBOR-Zuordnung (definiert als `Zeitstempel-Zuordnung`), die mindestens ein Schlüssel-Wert-Paar (definiert als `Zeitstempel-Eintrag`) enthalten muss. Der Schlüssel muss die hier definierte C2PA-Manifest-URN für das C2PA-Manifest sein, das mit einem Zeitstempel versehen wird, und der Wert muss eine CBOR-Byte-Zeichenkette sein, deren Inhalt im folgenden Absatz beschrieben wird.

Der Wert für jeden `Zeitstempel-Eintrag` muss denselben Binärdaten entsprechen, die im Feld `„timeStampToken”` der Struktur `„TimeStampResp”` enthalten sind, die als Antwort von einer RFC 3161-konformen Zeitstempelstelle (TSA) (RFC 3161) im Detached Content Mode empfangen wurde. Die `TimeStampResp` selbst muss unter Verwendung des gleichen Verfahrens wie in Abschnitt 10.3.2.5.3, „Erhalten des Zeitstempels“, beschrieben, mit der Ausnahme, dass der Wert der `Nutzlast` der Wert des Signaturfeldes der COSE_Sign1_Tagged-Struktur sein muss, die im C2PA-Claim-Signaturfeld des C2PA-Manifests enthalten ist, das mit einem Zeitstempel versehen wird.

18.18. Zertifikatsstatus

18.18.1. Beschreibung

In einigen Provenienz-Workflows wird ein Standard- oder Aktualisierungsmanifest offline erstellt, wobei es zum Zeitpunkt der Signierung nicht möglich ist, die Widerrufsinformationen (über OCSP) abzurufen. Wenn diese Informationen während des Validierungsprozesses nicht verfügbar sind, muss ein Validierer möglicherweise online gehen, um den Widerrufsstatus des Zertifikats zu ermitteln. Diese Assertion wird verwendet, um den vertrauenswürdigen Zertifikatsstatus für solche C2PA-Manifeste bereitzustellen, indem die Informationen nachträglich hinzugefügt werden.

18.18.2. Schema und Beispiel

Das Schema für diesen Typ wird durch die Regel `„cert-status-map”` in der folgenden CDDL-Definition definiert:

```
certificate-status-map = { „ocspVals”:
  [1* bstr]
}
```

Ein Beispiel im CBOR-Diagnoseformat (`.cbordiag`) ist unten aufgeführt:

```
{
  „ocspVals” : [ h'...',
    h'...'
  ]
}
```

18.18.3. Anforderungen

Eine Zertifikatsstatusangabe muss die Kennzeichnung „`c2pa.certificate-status`“ tragen, und ein C2PA-Manifest darf höchstens eine Zertifikatsstatusangabe enthalten.

Die Zertifikatsstatusangabe besteht aus einer CBOR-Zuordnung (definiert als `certificate-status-map`) und muss mindestens einen Eintrag im ocspVals-Array enthalten. Wie in Abschnitt 14.5.2, „Zertifikatswiderruf“, beschrieben, fragt der Anspruchgenerator den durch das Signaturzertifikat angegebenen OCSP-Dienst ab, erfasst die Antwort und speichert sie im gleichen Binärformat, das auch für die Speicherung als Element des ocspVals-Arrays des rVals-Headers verwendet wird (siehe Beispiel 3, „CDDL für rVals“).

18.19. Asset-Referenz

18.19.1. Beschreibung

Diese Angabe wird verwendet, um einen oder mehrere Orte anzugeben, an denen eine Kopie der Ressource erhältlich ist. Diese Orte sind jeweils mit einer Angabe zur Ressourcenreferenz zu beschreiben. Der Ort ist über eine URI anzugeben. Die URI kann entweder auf eine einzelne Ressource verweisen oder auf ein Verzeichnis. Im letzteren Fall dient sie dazu, den Ort für eine Sammlung von Ressourcen anzugeben, die über einen Sammlungsdaten-Hash gehasht werden.

HINWEIS	Die Angabe einer `URI` bietet Flexibilität bei der Beschaffung der Ressource aus Web-Standorten oder verteilten Dateisystemen wie oder IPFS (siehe https://docs.ipfs.tech/how-to/address-ipfs-on-web/#subdomain-gateway für Letzteres).

Eine Asset-Referenz-Assertion muss die Bezeichnung `c2pa.asset-ref` haben.

Der Zeitstempel in den Metadaten der Assertion dient als Grundlage für die Bestimmung der Aktualität des als Referenz beschriebenen Links.

18.19.2. Schema und Beispiel

Das Schema für diesen Typ wird durch die Regel „`asset-ref-map`” in der folgenden CDDL-Definition definiert:

```
Die Asset-Referenz-Assertion (ARA) beschreibt, wo eine Kopie des Assets erhältlich ist. asset-ref-map = {
  „references”: [1* ara-reference-block-map]
}
```

```
ara-reference-block-map = { „reference“: ara-
   reference-uri-map,
   ? „description”: tstr, ; Für Menschen lesbare Beschreibung des Standorts.
}

ara-reference-uri-map = {
   „uri”: tstr, ; URI, die auf einen Ort verweist, an dem eine Kopie des Assets erhältlich ist
}
```

Ein Beispiel in CBOR-Diagnoseschrift (RFC 8949, Abschnitt 8) ist unten aufgeführt:

```
{
   „references”: [
      {
        „description”: „Eine Kopie des Assets im Web”, „reference”: {
             „uri”: „https://some.storage.us/foo”
          }
      },
      {
        „Beschreibung“: „Eine Kopie der Ressource auf IPFS“,
        „Referenz“: {
             „uri“: „ipfs://cid“
          }
      }
   ]
}
```

18.20. Asset-Typ

18.20.1. Beschreibung

Die Asset-Typ-Aussage bietet eine Möglichkeit, ein Asset vollständiger zu beschreiben, insbesondere zusätzliche Informationen darüber, wie es zu analysieren oder anderweitig zu verarbeiten ist. Diese Aussage ermöglicht die Angabe eines IANA-Medientypwerts und/oder zusätzlicher Typinformationen, da viele Assets Formate haben, die nicht vollständig durch einen einzigen Medientypwert beschrieben werden können.

Die Asset-Typ-Assertion muss die Bezeichnung `„c2pa.asset-type.v2”` tragen. In einem C2PA-Manifest darf höchstens eine Asset-Typ-Assertion enthalten sein.

HINWEIS Frühere Versionen dieses Standards dokumentierten eine `c2pa.asset-type`-Assertion, die nun veraltet ist.

Wenn vorhanden, muss der Wert des Feldes `„dc:format“` dem IANA-Medientyp des Assets entsprechen.

Wenn vorhanden, muss der Wert des Feldes `„types“` ein Array aus null oder mehr Zuordnungen (`Asset-Typ-Zuordnung`) sein, die die mit dem Asset verbundenen Typen angeben. Der Wert des Feldes `„type“` in dieser Zuordnung muss entweder aus Tabelle 11, „Asset-Typ-Werte“, stammen oder einen entitätsspezifischen Namensraum (z. B. `com.litware.types.abc`) verwenden, der der in Abschnitt 6.2.2, „Benennung von Labels“, definierten Syntax für Assertion-Labels entspricht. Falls relevant, kann die Version des Assets (z. B. die Version eines Datensatzes oder Modells) im Feld `„version“` in der `asset-type-map` dokumentiert werden.

ANMERKUNG

Da C2PA eingeführt wird, um in Zukunft die Herkunft von KI-/ML-Assets (d. h. Assets für künstliche Intelligenz/maschinelles Lernen) anzugeben, kann das C2PA-Manifest in die Modell- und Datensatz-Assets eingebettet werden, und die Asset-Typ-Aussage kann verwendet werden, um den Typ dieser Modell- und Datensatz-Assets anzugeben.

Tabelle 11. Werte für den Asset-Typ

C2PA-Typ	Beschreibung des C2PA-Typs der Ressource
c2pa.types.dataset	AI/ML-Datensatz, der von mehreren AI/ML-Frameworks verarbeitet werden kann oder durch keinen anderen Wert beschrieben wird
c2pa.types.dataset.jax	JAX-Datensatz
c2pa.types.dataset.keras	Keras-Datensatz
c2pa.types.dataset.ml_net	ML.NET-Datensatz
c2pa.types.dataset.mxnet	MXNet-Datensatz
c2pa.types.dataset.onnx	ONNX-Datensatz
c2pa.types.dataset.openvino	OpenVINO-Datensatz
c2pa.types.dataset.pytorch	PyTorch-Datensatz
c2pa.types.dataset.tensorflow	TensorFlow-Datensatz
c2pa.types.model	KI/ML-Modell, das durch keinen anderen Modelltyp beschrieben wird
c2pa.types.model.jax	JAX-Modell
c2pa.types.model.keras	Keras-Modell
c2pa.types.model.ml_net	ML.NET-Modell
c2pa.types.model.mxnet	MXNet-Modell
c2pa.types.model.onnx	ONNX-Modell
c2pa.types.model.openvino.parameter	OpenVINO-Modellparameter
c2pa.types.model.openvino.topology	OpenVINO-Modelltopologie
c2pa.types.model.pytorch	PyTorch-Modell
c2pa.types.model.tensorflow	TensorFlow-Modell
c2pa.types.numpy	Gespeichert im serialisierten NumPy-Format
c2pa.types.protobuf	Gespeichert im Protocol Buffer-Format
c2pa.types.pickle	Gespeichert im Python-Pickle-Format
c2pa.types.savedmodel	Gespeichert im TensorFlow SavedModel-Format

18.20.2. Schema und Beispiel

Das Schema für diesen Typ wird durch die Regel „`asset-types`“ in der folgenden CDDL-Definition definiert:

```
; Die Asset-Typ-Aussage bietet eine Möglichkeit, den Typ oder das Format eines Assets zu beschreiben,
; insbesondere zusätzliche Informationen darüber, wie es zu parsen oder anderweitig zu verarbeiten ist.
; Sie kann auch verwendet werden, um extern referenzierte oder verwandte Assets wie AI/ML-Modelle zu
beschreiben.

$type-choice /= „c2pa.types.classifier“
$type-choice /= „c2pa.types.cluster“
$type-choice /= „c2pa.types.dataset”
$type-choice /= „c2pa.types.dataset.jax”
$type-choice /= "c2pa.types.dataset.keras"
$type-choice /= "c2pa.types.dataset.ml_net"
$type-choice /= "c2pa.types.dataset.mxnet"
$type-choice /= „c2pa.types.dataset.onnx”
$type-choice /= „c2pa.types.dataset.openvino”
$type-choice /= „c2pa.types.dataset.pytorch”
$type-choice /= „c2pa.types.dataset.tensorflow”
$type-choice /= „c2pa.types.format.numpy”
$type-choice /= „c2pa.types.format.protobuf”
$type-choice /= „c2pa.types.format.pickle”
$type-choice /= "c2pa.types.generator"
$type-choice /= „c2pa.types.generator.prompt”
$type-choice /= „c2pa.types.generator.seed”
$type-choice /= „c2pa.types.model”
$type-choice /= „c2pa.types.model.jax”
$type-choice /= „c2pa.types.model.keras”
$type-choice /= „c2pa.types.model.ml_net”
$type-choice /= „c2pa.types.model.mxnet”
$type-choice /= „c2pa.types.model.onnx”
$type-choice /= „c2pa.types.model.openvino”
$type-choice /= „c2pa.types.model.openvino.parameter”
$type-choice /= „c2pa.types.model.openvino.topology”
$type-choice /= „c2pa.types.model.pytorch”
$type-choice /= „c2pa.types.model.tensorflow”
$type-choice /= „c2pa.types.regressor”
$type-choice /= „c2pa.types.tensorflow.hubmodule”
$type-choice /= "c2pa.types.tensorflow.savedmodel"
$type-choice /= tstr .regexp "([\\da-zA-Z_-]+\\.)+[\\da-zA-Z_-]+"

asset-type-map = {
  "type": $type-choice, ; eine der aufgeführten Optionen oder ein benutzerdefinierter Wert
  ? "version": tstr .regexp "^(0|[1-9]\\d*)\\.(0|[1-9]\\d*)\\.(0|[1-9]\\d*)(?:-((?:0|[1-

]*))*))?(?:\\+([0-9a-zA-Z-]+(?:\\.[0-9a-zA-Z-]+)*))?$"
}

asset-types = {
  ? "dc:format": format-string, ; IANA-Medientyp des Assets
  ? "types": [* asset-type-map], ; eine Sammlung von Typen, die mit dem Asset in Zusammenhang stehen
  ? „metadata”: $assertion-metadata-map ; zusätzliche Informationen zur Assertion
}
```

Ein Beispiel in CBOR-Diagnoseschrift (RFC 8949, Abschnitt 8) ist unten aufgeführt. In diesem Beispiel handelt es sich bei der Ressource um eine TensorFlow-Modelldatei der Version 2.11.0, die im SavedModel-Format gespeichert ist.

```
{
  „types”:
  [
    {
      "type": "c2pa.types.model.tensorflow",
```

```
        „version": „2.11.0"
      },
      {
        „Typ": „c2pa.types.savedmodel", „Version":
        „2.11.0"
      }
    ]
}
```

18.20.3. Details zur Auswahl eines Werts für `den Typ`

Wenn der genaue Typ eines Assets im IANA-Registrierungsanwendungstyp oder im IANA-Registrierungstext-Typ angegeben ist, einschließlich JSON-, CSV- und XML-Typen, sollten diese Informationen im Feld `„dc:format"` der Asset-Typ-Aussage enthalten sein.

> Wenn es sich bei dem Asset beispielsweise um eine Textdatei im CSV-Format handelt, wäre das Feld `„dc:format"` `text/csv`.

Eine Asset-Typ-Angabe kann sowohl ein Dublin-Core-Format als auch einen C2PA-Standard oder einen benutzerdefinierten Asset-Typ enthalten, um zusätzliche Informationen über den Typ des Assets bereitzustellen. Einige bestehende Dublin-Core-Typen, die häufig in einer Asset-Typ-Angabe in Kombination mit anderen Asset-Typen verwendet werden, sind in Tabelle 12, „Gängige DC-Formate", aufgeführt.

Tabelle 12. Gängige DC-Formate

dc:format Wert	Beschreibung des Dublin-Core-Typs des Assets
application/json	Gespeichert im JSON-Format
application/gzip	Gespeichert im GZIP-Format
application/vnd.rar	Gespeichert im RAR-Format
application/zip	Gespeichert im ZIP-Format
application/octet-stream	Gespeichert in einem beliebigen Binärformat
text/csv	Gespeichert im CSV-Format
text/plain	Gespeichert im Klartextformat
text/tab-separated-values	Gespeichert im TSV-Textformat (Tab-Separated-Values)
text/xml	Gespeichert im XML-Format

IANA-strukturierte Suffixe wie `+json` und `+zip` werden ebenfalls im Feld `„dc:format"` des C2PA-Claims unterstützt, um zusätzliche Typen anzugeben.

Einige dc:format-Typen werden häufig verwendet, sind jedoch nicht im IANA-Register angegeben. Die folgenden `dc:format`
Die Werte gelten für C2PA-Assets, wie in Tabelle 13 „Zusätzliche Formate" dargestellt.

Tabelle 13. Zusätzliche Formate

dc:format Wert	Beschreibung des Dublin Core-Typs des Assets
application/x-hdf5	Gespeichert im HDF5-Format
application/x-7z-compressed	Gespeichert im 7Z-Format

18.21. Tiefenkarte

18.21.1. Beschreibung

Eine Tiefenkarten-Assertion liefert eine 3D-Beschreibung der von einer Kamera aufgenommenen Szene. Eine Tiefenkarten-Assertion kann eine vorab berechnete Tiefenkarte oder Daten enthalten, die später zur Berechnung einer Tiefenkarte durch nachgeschaltete Erfassungs- oder Anzeigesoftware verwendet werden können (z. B. linke/rechte Stereobilder).

Alle Tiefenkarten-Assertions müssen eine Kennzeichnung haben, die mit `c2pa.depthmap` beginnt und auf die ein dritter Abschnitt folgt, der den Typ der Tiefenkarte angibt.

C2PA-Tiefenkarten-Aussagen müssen optisch erfasst werden und dürfen nicht aus einem einzelnen 2D-Bild abgeleitet werden, beispielsweise mithilfe eines Modells für maschinelles Lernen.

18.21.2. GDepth-Tiefenkarte

Eine GDepth-Tiefenkarten-Assertion nutzt das etablierte GDepth-Format, um eine vorberechnete Tiefenkarte zu kodieren. Eine GDepth-

Tiefenkarten-Assertion muss die Bezeichnung `c2pa.depthmap.GDepth` tragen.

Das Schema für die in dieser Assertion gespeicherten Daten muss immer dem Schema unter https://developers.google.com/ depthmap-metadata/reference entsprechen.

HINWEIS Es gibt keine Bedenken hinsichtlich der Aufteilung der GDepth-Daten, wenn diese 64 KB überschreiten, da diese Grenze in XMP existierte, um den Beschränkungen der APP1-Segmentgröße Rechnung zu tragen.

18.21.3. Schema und Beispiel

Das Schema für diesen Typ wird durch die Regel `„depthmap-gdepth-map”` in der folgenden CDDL-Definition definiert:

```
; Assertion, die eine GDepth-formatierte 3D-Tiefenkarte der erfassten Szene codiert depthmap-
gdepth-map = {
  „GDepth:Format“: Format-Typ, ; Das Format, das beschreibt, wie die Tiefenkartendaten in eine gültige
Fließkomma-Tiefenkarte umgewandelt werden. Derzeit gültige Werte sind „RangeInverse“ und „RangeLinear“.
  „GDepth:Near“: Float, ; Der Nahwert der Tiefenkarte in Tiefeneinheiten „GDepth:Far“: Float,
  ; Der Fernwert der Tiefenkarte in Tiefeneinheiten
  „GDepth:Mime“: MIME-Typ, ; Der MIME-Typ für die Base64-Zeichenfolge, die den Inhalt des Tiefenbildes
beschreibt, z. B. „image/jpeg“ oder „image/png“
  „GDepth:Data”: Base64-Zeichenfolgen-Typ, ; Das Base64-kodierte Tiefenbild. Siehe Seite zur GDepth-Kodierung
unter developers.google.com. Die Tiefenkarte wird so gestreckt, dass sie in das entsprechende Farbbild passt.
  ? „GDepth:Units”: Einheitentyp,       ; Die Einheiten der Tiefenkarte, z. B. „m” für Meter oder „mm” für
Millimeter
```

```
  ? „GDepth:MeasureType”: depth-meas-type, ; Der Typ der Tiefenmessung. Derzeit gültige Werte sind „OpticalAxis”
und „OpticRay”.
  ? „GDepth:ConfidenceMime”: confidence-mime-type, ; Der MIME-Typ für die Base64-Zeichenfolge, die den Inhalt des
Konfidenzbildes beschreibt, z. B. „image/png”.",
  ? „GDepth:Confidence”: Base64-Zeichenfolge,            ; Das Base64-kodierte Konfidenzbild. Siehe die Seite
zur GDepth-Kodierung unter developers.google.com. Die Konfidenzkarte sollte dieselbe Größe wie die
Tiefenkarte haben
  ? „GDepth:Manufacturer”: tstr .size (1..max-tstr-length), ; Der Hersteller des Geräts, das diese
Tiefenkarte erstellt hat
  ? „GDepth:Model”: tstr .size (1..max-tstr-length), ; Das Modell des Geräts, das diese Tiefenkarte erstellt hat
  ? „GDepth:Software”: tstr .size (1..max-tstr-length), ; Die Software, mit der diese Tiefenkarte erstellt wurde
  ? „GDepth:ImageWidth”: float, ; Die Breite in Pixeln des Originalfarbbildes, das dieser Tiefenkarte
zugeordnet ist. Dies ist NICHT die Breite der Tiefenkarte. Falls vorhanden, müssen Anwendungen diese
Eigenschaft beim Skalieren, Zuschneiden oder Drehen des Farbbildes aktualisieren. Clients verwenden diese
Eigenschaft, um die Integrität der Tiefenkarte in Bezug auf das Farbbild zu überprüfen
  ? „GDepth:ImageHeight”: float, ; Die Höhe in Pixeln des Originalfarbbildes, das mit dieser Tiefenkarte
verbunden ist. Dies ist NICHT die Höhe der Tiefenkarte. Wenn vorhanden, müssen Apps diese Eigenschaft beim
Skalieren, Zuschneiden oder Drehen des Farbbildes aktualisieren. Clients verwenden diese Eigenschaft, um die
Integrität der Tiefenkarte in Bezug auf das Farbbild zu überprüfen.
  ? „metadata”: $assertion-metadata-map, ; zusätzliche Informationen zur Assertion
}

base64-string-type = tstr

$mime-choice /= „image/jpeg”
$mime-choice /= „image/png“

mime-type = $mime-choice .default „image/jpeg” confidence-mime-type
= $mime-choice .default „image/png”

$format-choice /= „RangeInverse”
$format-choice /= "RangeLinear"

format-type = $format-choice .default "RangeInverse"

; Die Einheit kann Meter (dargestellt als „m“) oder Millimeter (dargestellt als „mm“) sein.
$unit-choice /= „m“
$unit-choice /= „mm“
unit-type = $unit-choice .default "m"

$depth-meas-choice /= „OpticalAxis”
$depth-meas-choice /= „OpticRay”
depth-meas-type = $depth-meas-choice .default "OpticalAxis"
```

Ein Beispiel in CBOR-Diagnoseschrift (RFC 8949, Abschnitt 8) ist unten aufgeführt:

```
{
  „GDepth:Far”: 878,7,
  „GDepth:Data”: „hoOspQQ1lFTy/4Tp8Epx670E5QW5NwkNR+2b30KFXug=”, „GDepth:Mime”:
  „image/jpeg”,
  „GDepth:Near”: 29,3,
  „GDepth:Model”: „CameraCompany Shooter S1”, „GDepth:Units”: „mm”,
  „GDepth:Format“: „RangeInverse“,
  "GDepth:Software": "Truepic Foresight Firmware für QC QRD8250 v0.01", "GDepth:Confidence":
  "acdbpQQ1lFTy/4Tp8Epx670E5QW5NwkNR+2b30KFXug=", "GDepth:ImageWidth": 32.2,
  „GDepth:ImageHeight”: 43,6
```

```
  „GDepth:MeasureType”: „OpticalAxis”,
  „GDepth:Manufacturer”: „CameraCompany”,
  „GDepth:ConfidenceMime”: „image/png”,
}
```

Gemäß der GDepth-Spezifikation müssen die folgenden Felder in allen GDepth-Tiefenkarten-Assertions vorhanden sein:

- GDepth:Format;
- GDepth:Near;
- GDepth:Far;
- GDepth:Mime;
- GDepth:Data.

18.22. Schriftartinformationen

18.22.1. Beschreibung

Eine Font-Information-Assertion wird verwendet, um sicherzustellen, dass grundlegende Font-Metadaten wie Name, Format, Urheberangabe und Lizenzierung dem Asset in einer Weise hinzugefügt werden, die kryptografisch validiert werden kann.

Eine Font-Information-Assertion muss die Bezeichnung „`font.info`“ tragen, und es darf höchstens eine Font-Information-Assertion pro Manifest vorhanden sein.

18.22.2. Schema und Beispiel

Das Schema für diesen Typ wird durch die Regel „`font-info-map`” in der folgenden CDDL-Definition definiert:

```
; Assertion-Daten für die font.info-Assertion.
font-info-map = {
  „fullName”: tstr, ; Der vollständige Name der Schriftart.
  ; Eine Version im Semantic Versioning (Semver)-Format.
  ? „version”: tstr .regexp „^(0|[1-9]\\d*)\\.(0|[1-9]\\d*)\\.(0|[1-9]\\d*)(?:-((?:0|[1-

]*))*))?(?:\\+([0-9a-zA-Z-]+(?:\\.[0-9a-zA-Z-]+)*))?$",
  ? "versionUrl": ext-url-type, ; Eine URL zu den Versionshinweisen zu dieser Version der Schriftart.
  ? „releaseDate”: tdate, ; Das Datum, an dem diese Version der Schriftart veröffentlicht wurde. „familyName”:
  tstr, ; Die Schriftfamilie.
  „style”: $font-style, ; Der Stil der Schriftart, z. B. kursiv oder normal. „weight”: font-
  weight-map, ; Die Schriftstärke mit Name und Wert.
  ; Der PostScript-Name, ID 6, aus der Schriftart-Namens-Tabelle.
  „postScriptName”: tstr .regexp „^(?!.*[\\[\\]\\(\\)\\{\\}<>\\/%])[!-~]{1,63}$”, ; Zeichen aus ASCII 33-126
mit Ausnahme der folgenden: [](){}<>/%
  „format“: $font-format-choice, ; Das Format dieser Schriftart. „copyrightNotice“:
  tstr, ; Das mit dieser Schriftart verbundene Urheberrecht.
  ? „copyrightHolder”: font-entity-map, ; Die Einheit, die das Urheberrecht an der Schriftart besitzt.
  ? „copyrightYears”: [1* font-copyright-year-range], ; Die Jahre, für die der Inhaber das Urheberrecht
geltend macht.
  ? „designers“: [1* font-designer-map], ; Die Personen, die die Schriftart entworfen haben.
 ? „designFoundry”: font-entity-map, ; Die Schriftgießerei, die die Schriftart entworfen hat.
```

```
  ? „sourceFoundry”: font-entity-map, ; Die Schriftgießerei, die die Schriftart vertreibt.
  ? „identifier“: tstr, ; Interne Kennung der Schriftart für die Verwendung durch die Schriftgießerei oder den
  Anbieter.
}

; Schriftartenformate
$font-format-choice /= „TrueType”
$font-format-choice /= „OpenType”

; Copyright-Jahresbereich
font-copyright-year-range = 1..9999

; Schriftstärkenbereich
font-weight-range = 1..1000

; Beschreibungen der Schriftstärkenklassen
$font-weight-class /= "Microline"
$font-weight-class /= „Hairline”
$font-weight-class /= „UltraThin”
$font-weight-class /= „ExtraThin”
$font-weight-class /= „Thin”
$font-weight-class /= „UltraLight”
$font-weight-class /= „ExtraLight”
$font-weight-class /= „Light”
$font-weight-class /= „SemiLight”
$font-weight-class /= „Book“
$font-weight-class /= „Normal”
$font-weight-class /= „Regular”
$font-weight-class /= "Medium"
$font-weight-class /= „DemiBold”
$font-weight-class /= „SemiBold”
$font-weight-class /= „Bold”
$font-weight-class /= „Heavy”
$font-weight-class /= „ExtraBold”
$font-weight-class /= „UltraBold”
$font-weight-class /= „SemiBlack”
$font-weight-class /= „Black”
$font-weight-class /= „ExtraBlack”
$font-weight-class /= „UltraBlack”
$font-weight-class /= „MegaBlack”

; Der Schriftstil
$font-style /= "Normal"
$font-style /= "Italic"
$font-style /= „Oblique”
$font-style /= „Roman”
$font-style /= „Regular“

; Daten für eine
Schriftstärke font-weight-map
= {
  "class": $font-weight-class, ; Der beschreibende Name der Gewichtsklasse, z. B. fett oder dünn.
  „Wert“: font-weight-range, ; Der Wert der Schriftstärke.
}

; Daten für eine Entität mit einem Namen und Anmeldedaten
font-entity-map = {
  „name“: tstr, ; Der Name der Person oder Gießerei.
  ? „url“: ext-url-type, ; Eine URL für zusätzliche Informationen zu dieser Person oder Schriftgießerei.
}

; Daten für einen
Schriftdesigner font-designer-
map = {
  „person”: font-entity-map, ; Die Person, die die Schriftart entworfen hat.
```

```
  ? „foundry“: font-entity-map, ; Der Name der Schriftgießerei, mit der der Designer bei der Gestaltung der
Schrift verbunden war.
  ? „Beitrag“: tstr, ; Eine Beschreibung dessen, was der Designer zum Font beigetragen hat. Zum Beispiel „Alle
lateinischen und arabischen Zeichen“.
  ? „startDate“: tdate, ; „Wann der Designer mit der Arbeit an der Schrift begonnen hat.
  ? „endDate“: tdate, ; Wann der Designer seine Beiträge zum Schriftdesign beendet hat.
}
```

Ein einfaches Beispiel in CBOR-Diagnoseschrift (RFC 8949, Abschnitt 8), das nur erforderliche Felder enthält, ist unten aufgeführt:

```
{
     „fullName”: „Beispiel Zwei Kursivschrift”,
     „familyName”: „BeispielZwei”, „style”:
     „Kursivschrift”,
     „weight”: {
          "class": "Regular", "value":
          400
     },
     „postScriptName“: „Beispiel-Zwei-Kursiv“,
     „format“: „TrueType“,
     „copyrightNotice“: „Copyright 2011 Die Autoren des Beispiel Zwei-Projekts
(https://www.example.com/lifonts/Example-Two), mit reserviertem Schriftnamen ‚Beispiel Zwei‘“,
     „copyrightHolder“: { „name“:
          „Fabrikam“
     },
     „designer”: [
          {
               „person“: {
                  "name": "John Doe",
                  „url“: „https://fabrikam.example.com/jdoefonts“
               }
          }
     ]
}
```

Dieses erweiterte Beispiel zeigt auch optionale Felder:

```
{
     „fullName”: „Beispielschriftart fett kursiv”,
     „version”: „7.0.4-beta”,
     „versionUrl”: „https://fabrikam.example.com/release/efbi/7.0”, „familyName”:
     „ExampleFont”,
     „style”: „Italic”,
     „weight”: {
          "class": "Bold",
          "value": 700
     },
     „postScriptName“: „Beispielschrift-Fettkursiv“, „format“:
     „OpenType“,
     „copyrightNotice”: „© 2017 Fabrikam, Inc. Alle Rechte vorbehalten.”,
     „copyrightHolder”: {
          "name": "Fabrikam Inc."
     },
     „copyrightYears“: [
          1982,
          2017
     ],
     „designer“: [
```

```
          {
               „person“: {
                  „name“: „John Doe“,
                  „url“: „https://fabrikam.example.com/browse/designers/john-doe“
               },
               „Foundry“: {
                   "name": "Fabrikam Fonts"
               },
               „Beitrag“: „Ligaturen“.
          },
          {
               „person“: {
                  „name“: „Jane Doe“
               },
               „Schriftgießerei“: {
                    „name“: „Fabrikam Fonts“
               },
               „Beitrag“: „Alle Zeichen.“
          }
     ],
     „designFoundry“: {
          "name": "Fabrikam Fonts",
          „url“: „https://fabrikam.example.com“
     },
     „sourceFoundry“: {
          „name“: „Fonts Direct 2 U“, „url“:
          „https://fd2u.example.com“
     },
     „identifier”: „ExampleFont Bold Italic (Fabrikam)”
}
```

Kapitel 19. Patentpolitik

Die C2PA hat über den Patentmodus des W3C (2004) eine offene Standard-Patentpolitik verabschiedet:

Lizenzverpflichtung. Für Materialien, die nicht zum Quellcode oder zu den Datensätzen gehören, die von der Arbeitsgruppe entwickelt wurden, verpflichtet sich jeder Teilnehmer der Arbeitsgruppe, alle seine wesentlichen Ansprüche gemäß der Definition in der W3C-Patentrichtlinie (verfügbar unter http://www.w3.org/Consortium/Patent-Policy-20040205) gemäß den W3C-RF-Lizenzanforderungen in Abschnitt 5 (http://www.w3.org/Consortium/Patent-Policy-20040205) in den von dieser Arbeitsgruppe verabschiedeten genehmigten Ergebnissen zur Verfügung zu stellen, als ob diese genehmigten Ergebnisse eine W3C-Empfehlung wären. Der von der Arbeitsgruppe entwickelte Quellcode unterliegt der in der Arbeitsgruppen-Charta festgelegten Lizenz.

Ausschluss. Vor der Annahme eines Entwurfs als genehmigtes Ergebnis kann ein Teilnehmer der Arbeitsgruppe wesentliche Ansprüche aus seinen Lizenzverpflichtungen im Rahmen dieser Vereinbarung ausschließen, indem er dem Vorsitzenden der Arbeitsgruppe eine schriftliche Mitteilung über diese Absicht zukommen lässt („Ausschlussmitteilung“). Die Ausschlussmitteilung für erteilte Patente und veröffentlichte Anmeldungen muss die Patentnummer(n) oder den Titel und die Anmeldungsnummer(n) für jedes der erteilten Patente oder anhängigen Patentanmeldungen enthalten, die der Teilnehmer der Arbeitsgruppe von der in Abschnitt 1 dieser Patentrichtlinie festgelegten Lizenzverpflichtung ausschließen möchte. Wenn ein erteiltes Patent oder eine anhängige Patentanmeldung, die wesentliche Ansprüche enthalten kann, nicht in der Ausschlusserklärung aufgeführt ist, unterliegen diese wesentlichen Ansprüche weiterhin den Lizenzverpflichtungen gemäß dieser Vereinbarung. Die Ausschlussmitteilung für unveröffentlichte Patentanmeldungen muss entweder (i) den Wortlaut der eingereichten Anmeldung oder (ii) die Angabe der spezifischen Teile des Entwurfs des Liefergegenstands enthalten, deren Umsetzung den ausgeschlossenen Anspruch zu einem wesentlichen Anspruch macht. Wenn (ii) gewählt wird, beschränkt sich die Wirkung des Ausschlusses auf die angegebenen Teile des Entwurfs des Liefergegenstands. Der Vorsitzende der Arbeitsgruppe veröffentlicht die Ausschlussmitteilungen.

Anhang A: Einbettung von Manifesten

A.1. Unterstützte Formate

Ein C2PA-Manifest wird als Teil des C2PA-Manifest-Speichers für dieses Asset in ein Asset eingebettet.

Bei der Einbettung des C2PA-Manifest-Speichers in ein Asset variiert der Speicherort je nach Typ oder Format des Assets. Hier sind einige bekannte Dateiformate und der Speicherort für den C2PA-Manifest-Speicher in jedem dieser Formate:

JPEG

Weitere Informationen finden Sie in Abschnitt A.3.1, „Einbetten von Manifesten in JPEG“.

JPEG-XL

Weitere Informationen finden Sie in Abschnitt A.3.8, „Einbetten von Manifesten in JPEG XL“.

PNG

Weitere Informationen finden Sie in Abschnitt A.3.2, „Einbetten von Manifesten in PNG“.

SVG

Weitere Informationen finden Sie in Abschnitt A.3.3, „Einbetten von Manifesten in SVG“.

FLAC

Weitere Informationen finden Sie in Abschnitt A.3.4, „Einbetten von Manifesten in ID3“.

MP3

Weitere Informationen finden Sie in Abschnitt A.3.4, „Einbetten von Manifesten in ID3“.

GIF

Weitere Informationen finden Sie in Abschnitt A.3.7, „Einbetten von Manifesten in GIFs“.

DNG

Weitere Informationen finden Sie in Abschnitt A.3.5, „Einbetten von Manifesten in TIFF-basierte Assets“.

TIFF-basierte Formate

Weitere Informationen finden Sie in Abschnitt A.3.5, „Einbetten von Manifesten in TIFF-basierte Assets“.

WAV und BWF

Weitere Informationen finden Sie in Abschnitt A.3.6, „Einbetten von Manifesten in RIFF-basierte Assets“.

AVI

Weitere Informationen finden Sie in Abschnitt A.3.6, „Einbetten von Manifesten in RIFF-basierte Assets“.

WebP

Weitere Informationen finden Sie in Abschnitt A.3.6, „Einbetten von Manifesten in RIFF-basierte Assets“.

Andere RIFF-basierte Formate

Weitere Informationen finden Sie in Abschnitt A.3.6, „Einbetten von Manifesten in RIFF-basierte Assets“.

Schriftarten

Weitere Informationen finden Sie in Abschnitt A.3.9, „Einbetten von Manifesten in Schriftarten“.

PDF

Weitere Informationen finden Sie in Abschnitt A.4, „Manifeste in PDF-Dateien einbetten“.

EPUB

Weitere Informationen finden Sie in Abschnitt A.6, „Einbetten von Manifesten in ZIP-basierte Formate“.

OOXML

Weitere Informationen finden Sie in Abschnitt A.6, „Einbetten von Manifesten in ZIP-basierte Formate“.

Open Document

Weitere Informationen finden Sie in Abschnitt A.6, „Einbetten von Manifesten in ZIP-basierte Formate“.

OpenXPS

Weitere Informationen finden Sie in Abschnitt A.6, „Einbetten von Manifesten in ZIP-basierte Formate“.

Andere ZIP-basierte Formate

Weitere Informationen finden Sie in Abschnitt A.6, „Einbetten von Manifesten in ZIP-basierte Formate“.

MP4

Weitere Informationen finden Sie in Abschnitt A.5, „Einbetten von Manifesten in BMFF-basierte Assets“.

MOV

Weitere Informationen finden Sie in Abschnitt A.5, „Einbetten von Manifesten in BMFF-basierte Assets“.

AAC

Weitere Informationen finden Sie in Abschnitt A.5, „Einbetten von Manifesten in BMFF-basierte Assets“.

ALAC

Weitere Informationen finden Sie in Abschnitt A.5, „Einbetten von Manifesten in BMFF-basierte Assets“.

HEIF

Weitere Informationen finden Sie in Abschnitt A.5, „Einbetten von Manifesten in BMFF-basierte Assets“.

Andere BMFF-basierte Formate

Das in Abschnitt A.5, „Einbetten von Manifesten in BMFF-basierte Assets“, angegebene Feld.

HINWEIS

Zu den nicht BMFF-basierten Audioformaten, deren Aufnahme in diese Spezifikation in Betracht gezogen wird, gehören Ogg Vorbis und die native Container-Version des Free Lossless Audio Codec (Native FLAC).

A.2. Einbetten von Manifesten in mehrteilige Assets

Bei der Einbettung eines C2PA-Manifests in ein mehrteiliges Asset („Multi-Asset“) muss ein C2PA-Manifest-Speicher in den primären Teil des Assets (der das aktive Manifest enthält) eingebettet sein, wobei zusätzliche Teile auch eigene C2PA-Manifest-Speicher enthalten können. Das aktive Manifest des Hauptteils muss eine Multi-Asset-Hash-Assertion enthalten, die den Speicherort und den Hash jedes Teils innerhalb des Assets beschreibt und die Herkunft des gesamten mehrteiligen Assets beschreiben sollte.

A.3. Einbetten von Manifesten in nicht BMFF-basierte Assets

A.3.1. Einbetten von Manifesten in JPEG

Der C2PA-Manifest-Speicher muss als Daten eingebettet werden, die in einem **APP1**1-Markersegment gemäß der Definition in JPEG XT, ISO/IEC 18477-3 enthalten sind.

Da ein einzelnes Markersegment in JPEG 1 nicht größer als 64 KB sein darf, sind wahrscheinlich mehrere APP11-Segmente erforderlich, die gemäß dem JPEG 1-Standard und ISO 19566-5:2023, D.2 aufgebaut sein müssen. Beim Schreiben mehrerer Segmente müssen diese in sequenzieller Reihenfolge geschrieben werden und sie müssen zusammenhängend sein (d. h. ein Segment muss unmittelbar auf das nächste folgen).

A.3.2. Einbetten von Manifesten in PNG

Der C2PA Manifest Store muss mithilfe eines zusätzlichen, privaten, nicht kopierbaren Chunks vom Typ „caBX“ (gemäß PNG, 4.7.2) eingebettet werden. Es wird empfohlen, dass der „caBX”-Chunk vor den „IDAT”-Chunks steht.

HINWEIS

Obwohl PNG dies unterstützt, gilt es als schlechte Form, einen Datenblock nach dem „IDAT“ und vor dem „IEND“ zu haben. (Ausgenommen sind animierte PNG-Blöcke.)

A.3.3. Einbetten von Manifesten in SVG

SVG ist ein XML-basiertes Format, das entweder eigenständig oder eingebettet in andere textbasierte Formate wie HTML existieren kann. Daher ist es notwendig, den binären C2PA Manifest Store mit Base64 zu kodieren, um die Einbettung durchzuführen. In diesem Abschnitt wird beschrieben, wie dies zu tun ist, jedoch wird die Verwendung eines externen Manifests bevorzugt.

Der C2PA Manifest Store muss als Base64-codierter Wert eines `c2pa:`manifest-Elements in das Metadatenelement des SVG eingebettet werden. Da XML und insbesondere SVG dringend empfehlen, vor der Verwendung einen Namespace zu deklarieren, sollte dem svg-Element eine Attributdeklaration `xmlns:c2pa = „http://c2pa.org/manifest“` hinzugefügt werden.

Beispiel 17. Beispiel für einen C2PA-Manifest-Speicher in einer SVG-Datei

Ein Beispiel für einen C2PA-Manifest-Speicher in einer SVG-Datei (die tatsächlichen Daten des C2PA-Manifests wurden weggelassen).

```
<?xml version="1.0" standalone="yes"?>
<svg width="4in" height="3in" version="1.1" xmlns =
     "http://www.w3.org/2000/svg" xmlns:c2pa =
     "http://c2pa.org/manifest">
     <metadata>
          <c2pa:manifest>...Base64-Daten kommen hierhin...</c2pa:manifest>
     </metadata>
</svg>
```

A.3.4. Einbetten von Manifesten in ID3

Der C2PA-Manifest-Speicher muss in eine ID3v2-kompatible, komprimierte Audiodatei (z. B. MP3 oder FLAC) als gekapselte Objektdaten eines General Encapsulated Object (GEOB) eingebettet werden, wie in https://id3.org/id3v2.3.0 definiert. `Das MIME`-Typ-Feld des GEOB muss vorhanden sein und den Wert für den Medientyp für JUMBF verwenden, wie in Abschnitt 11.4, „Externe Manifeste“, beschrieben.

A.3.5. Einbetten von Manifesten in TIFF-basierte Assets

Das Digital Negative- oder DNG-Format ermöglicht es Kameraherstellern, ihre Kamera-Rohformate in standardisierter Form bereitzustellen. DNG basiert auf TIFF/EP (das wiederum auf TIFF basiert).

Der C2PA Manifest Store muss in eine TIFF-kompatible Datei (d. h. TIFF/EP, DNG oder andere TIFF-basierte RAW-Formate) als Daten eines Tags mit der ID 52545 (dezimal) oder 0xCD41 (hexadezimal) und dem Tag-Typ 7 eingebettet werden.

Obwohl TIFF das Konzept mehrerer Seiten oder Ebenen (über mehrere IFDs) unterstützt, darf es nur einen C2PA-Manifest-Speicher für das gesamte Asset geben – nicht einen pro IFD. Daher muss der C2PA-Manifest-Speicher das einzige Feld im letzten IFD sein, dem IFD unmittelbar vor dem Ende der Datei.

HINWEIS Frühere Versionen dieser Spezifikation erforderten die Verwendung von IFD 0, aber es wurde erkannt, dass dies die Verwendung in TIFF-basierten RAW-Formaten einschränkte.

A.3.6. Einbetten von Manifesten in RIFF-basierte Assets

Das RIFF-Format (Resource Interchange File Format) bietet ein generisches Containerformat zum Speichern von Daten in getaggten Blöcken. Es wird in erster Linie zum Speichern von Multimedia-Inhalten wie Bildern, Ton und Videos verwendet. Es dient als Containerformat für WAV, BWF, Broadcast Wave, AVI und WebP.

HINWEIS RIFF basiert auf einem älteren Format namens IFF.

Der C2PA-Manifest-Speicher wird als Daten eines Chunks mit der Kennung `C2PA` in eine RIFF-kompatible Datei (d. h. WAV, AVI oder WebP) eingebettet. Aus Kompatibilitätsgründen muss dieser C2PA-Chunk am Ende des RIFF-Chunks erscheinen.

A.3.7. Einbetten von Manifesten in GIFs

Der C2PA-Manifest-Speicher muss in Chunks mit einer Größe von maximal 255 Byte aufgeteilt und in zusammenhängende Daten-Subblöcke (gemäß GIF, 15) innerhalb eines C2PA-spezifischen Anwendungserweiterungsblocks (gemäß GIF, 26) eingebettet werden, wie unten angegeben.

HINWEIS In diesem C2PA-Anwendungserweiterungsblock wird der Anwendungsauthentifizierungscode nicht verwendet, um die Anwendung authentifizieren, die den Block erzeugt. Stattdessen wird es als Blockversion verwendet und zunächst auf Hauptversion 1, Nebenversion 0 gesetzt und wie unten angegeben codiert.

```
Erweiterungs-Einführung: 0x21
Anwendungserweiterungs-Bezeichnung: 0xFF
Blockgröße: 0xB
Anwendungsbezeichner: 0x43, 0x32, 0x50, 0x41, 0x5F, 0x47, 0x49, 0x46 („C2PA_GIF“)
Anwendungsauthentifizierungscode: 0x010000 (0x[Hauptversion][Nebenversion]00) Anwendungsdaten: Der C2PA-
Manifest-Speicher, codiert als eine Reihe von Daten-Unterblöcken, die jeweils 1 Byte groß sind und auf
die bis zu 255 Byte Daten folgen
Block-Terminator: 0x00 (wird nach dem letzten Daten-Unterblock des C2PA-Manifest-Speichers hinzugefügt) Anzahl:
Eins
```

Dieser Block muss nach dem Header und vor dem ersten Bilddeskriptor-Feld eingebettet werden.

A.3.8. Einbetten von Manifesten in JPEG XL

Wie in ISO/IEC 18181-2:2024, Abschnitt 4 beschrieben, unterstützt JPEG XL zwei verschiedene Formate für die Daten. Es kann eine mit JPEG 2000 und JPEG XS kompatible Box-Struktur verwenden oder es kann sich um einen direkten JPEG XL-Codestream ohne Box-Struktur handeln. Eine JPEG XL-Datei, die die Box-Struktur verwendet, darf höchstens eine JUMBF (`Jumb`)-Superbox (ISO/IEC 18181-2:2024, Abschnitt 9.3) enthalten, die eine C2PA-Manifest-JUMBF-Box enthält, welche das C2PA-Manifest gemäß Abschnitt 11.1.4.2, „Manifest Store“, enthält. Eine JPEG XL-Datei, die nur ein Codestream ist, kann kein eingebettetes C2PA-Manifest enthalten.

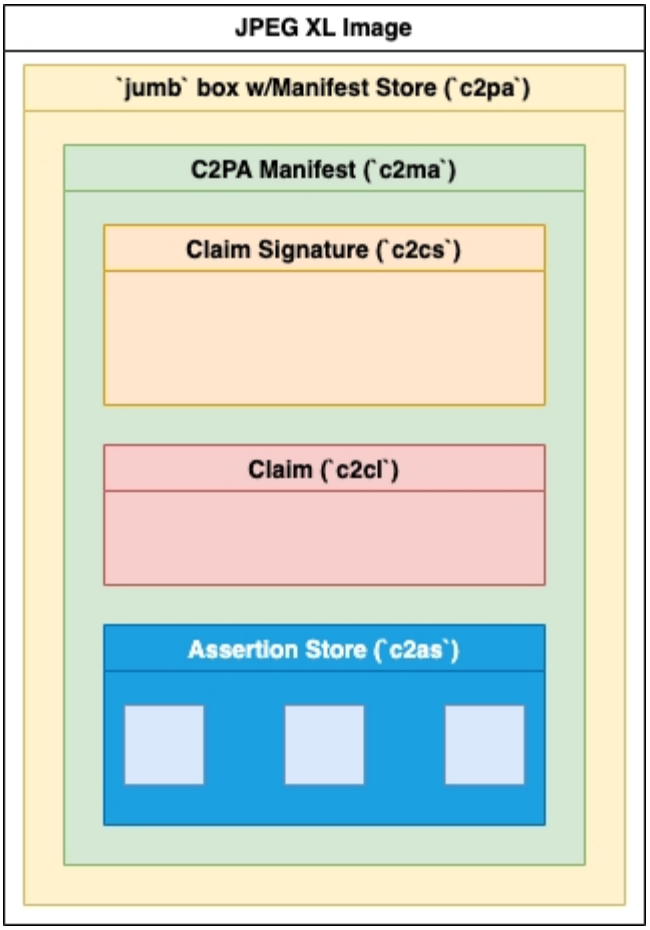

Abbildung 19. Ein in ein JPEG XL-Bild eingebettetes C2PA-Manifest

A.3.9. Einbetten von Manifesten in Schriftarten

Schriftarten, die entweder dem Open Font Format oder der OpenType-Spezifikation entsprechen, können eine C2PA-Tabelle enthalten. Wenn vorhanden, kann diese Tabelle ein eingebettetes Manifest, eine Remote-Manifest-URI oder beides enthalten.

Das Format `der C2PA`-Tabelle ist weder im Open Font Format noch in der OpenType-Spezifikation definiert; die folgende Definition ist vorläufig:

A.3.9.1. Tabellen-Tag

Der C2PA-Tabelleneintrag wird durch den folgenden Tabellen-Tag gekennzeichnet: `C2PA`.

A.3.9.2. Tabellen-Datensatz

Die C2PA-Tabelle bietet vollständige Unterstützung für Manifest-Speicher, die entweder eingebettet oder remote oder beides sein können. Der Tabelleneintrag ist wie folgt definiert:

Tabelle 14. C2PA-Tabelleneintrag

Typ	Name	Beschreibung
`uint16`	`majorVersion`	Gibt die Hauptversion der C2PA-Schriftartentabelle an.

uint16	minorVersion	Gibt die Nebenversion der C2PA-Schriftartentabelle an.
Offset32	activeManifestUriOffset	Offset vom Anfang der C2PA-Schriftartentabelle bis zum Abschnitt, der eine URI zum aktiven Manifest enthält. Wenn keine URI angegeben ist, sollte ein NULL-Offset = 0x0000 verwendet werden.
uint16	activeManifestUriLength	Länge der URI in Byte.
uint16	reserved	Für zukünftige Verwendung reserviert.
Offset32	manifestStoreOffset	Offset vom Anfang der C2PA-Fonttabelle bis zum Abschnitt, der einen C2PA-Manifest-Speicher enthält. Wenn kein Manifest-Speicher vorhanden ist, sollte ein NULL-Offset = 0x0000 verwendet werden.
uint32	manifestStoreLength	Länge der C2PA-Manifest-Speicherdaten in Byte.

Das nicht eingebettete C2PA-Manifest kann sich remote oder lokal auf demselben Speichersystem befinden. Wenn es sich bei der Referenz um eine JUMBF-URI handelt, sollte sie eine gültige Referenz innerhalb des C2PA-Manifestspeichers sein.

A.4. Einbetten von Manifesten in PDFs

A.4.1. Allgemeines

Alle C2PA-Manifest-Speicher müssen mithilfe eingebetteter Dateistreams (ISO 32000, 7.11.4) eingebettet werden. Das Wörterbuch für eingebettete Dateispezifikationen muss einen `Subtyp`-Schlüssel mit dem Wert „`application/c2pa`“ und einen `AFRelationship`-Schlüssel (ISO 32000, 7.11.3) mit dem Wert „`C2PA_Manifest`“ enthalten. Wenn ein C2PA-Manifest-Speicher in eine verschlüsselte PDF-Datei eingebettet ist, muss der eingebettete Dateistream einen Identitäts-Kryptografie-Filter verwenden.

A.4.2. Manifeste auf Dokumentebene

A.4.2.1. Hinzufügen des Manifests zu einer PDF-Datei

Beim Hinzufügen eines C2PA-Manifests zur gesamten PDF-Datei muss das Dokumentkatalogwörterbuch einen AF-Eintrag enthalten, dessen Wert ein indirekter Verweis auf die eingebettete Dateispezifikation ist, die das aktive Manifest enthält. Auf diese eingebettete Dateispezifikation muss ebenfalls über ein indirektes Objekt entweder aus dem `EmbeddedFiles` NameTree (`/Catalog/Names/EmbeddedFiles`) oder aus einer FileAttachment-Anmerkung verwiesen werden. Der Anmerkungsansatz muss verwendet werden, wenn ein C2PA-Manifest-Speicher zu einer PDF-Datei hinzugefügt wird, die bereits über eine bestehende PDF-Zertifizierungssignatur verfügt, um zu vermeiden, dass deren DocMDP-Einschränkungen ungültig werden.

HINWEIS Die Werte *1* oder *2* des P-Feldes im DocMDP-Wörterbuch lassen diese Art der Änderung nicht zu. Nur der Wert *3* ist zulässig.

In den meisten anderen Formaten gibt es nur einen einzigen C2PA-Manifest-Speicher, der alle C2PA-Manifeste für die Ressource enthält. Aufgrund der „inkrementellen Aktualisierungsfunktion” von PDF ist es jedoch erforderlich, stattdessen mehrere Manifeste in einer einzigen PDF-Datei zu unterstützen. In diesem Szenario gilt der C2PA-Manifest-Speicher in der Basis-PDF-Datei als ursprüngliches Manifest und derjenige in der letzten Aktualisierung als aktives Manifest. Ein C2PA-Manifest-Verbraucher muss alle C2PA-Manifeste in allen C2PA-Manifest-Speichern so verarbeiten, als wären sie in einem einzigen C2PA-Manifest-Speicher enthalten.

HINWEIS

Da eine JUMBF-URI immer eine vollständige URI ist, beginnt sie bei einem bestimmten C2PA-Manifest, und alle C2PA-Manifeste als in einem einzigen C2PA-Manifest-Speicher enthalten betrachtet werden, ist die Verwendung einer solchen URI zur Verweisung auf ein parentOf-Element über C2PA-Manifest-Speicher hinweg in einer PDF-Datei zulässig.

A.4.2.2. Kompatibilität mit PDF-Signaturen

Beim Hinzufügen eines neuen C2PA-Manifest-Speichers muss bekannt sein, ob auch eine PDF-Signatur (Zertifizierung oder Genehmigung) angewendet wird. Da die PDF-Signatur die Daten der PDF-Datei nach der Signierung des C2PA-Manifests verändert, müssen Größe und Speicherort des Contents-Schlüssels des PDF-Signaturwörterbuchs vor der C2PA-Signierung festgelegt werden. Dieser Bytebereich muss zur Liste der Ausschlüsse in der `c2pa.hash.` data-Assertion hinzugefügt werden, damit die C2PA-Signatur durch das Hinzufügen der PDF-Signatur nicht ungültig wird. Die PDF-Signatur muss sich über die gesamte PDF-Datei erstrecken, einschließlich des zugehörigen C2PA-Manifest-Speichers.

HINWEIS

Das Hinzufügen der PDF-Signatur zusätzlich zur C2PA-Claim-Signatur verbessert die Kompatibilität mit dem bestehenden PDF-Ökosystem.

A.4.3. Manifeste auf Objektebene

Zusätzlich zur Möglichkeit, die Herkunft der PDF-Datei selbst über Manifeste auf Dokumentebene anzugeben, können einzelne Objekte innerhalb eines Dokuments auch mit einem zugehörigen C2PA-Manifest-Speicher verknüpft werden. Dazu wird dem Stream oder Wörterbuch des Objekts ein AF-Eintrag hinzugefügt. Der Wert des AF-Eintrags muss ein indirekter Verweis auf die eingebettete Dateispezifikation sein, die den C2PA-Manifest-Speicher enthält, wie oben beschrieben.

Diese Funktion wird am häufigsten verwendet, um die Herkunft eingebetteter Bilder anzugeben – entweder als Bild- oder Formular-XObjects und Schriftarten. Sie kann auch verwendet werden, um die Herkunft bestimmter Inhalte anzugeben, indem der AF-Eintrag zum Objekt (über die Eigenschaftsliste) oder zu einem Strukturelement hinzugefügt wird, wie in der Klausel „Associated Files” (Zugehörige Dateien) der ISO 32000-2 (14.13.1) beschrieben.

Es wird empfohlen, jedes hinzugefügte Manifest auf Objektebene aus dem aktiven Manifest als componentOf-Bestandteil zu referenzieren. Auf diese Weise kann der C2PA-Manifest-Verbraucher die gesamte Herkunftskette des Assets leicht durchlaufen.

Im Allgemeinen kann jedem PDF-Stream oder Wörterbuch ein C2PA-Manifest beigefügt werden, sofern der Stream oder das Wörterbuch eine tatsächliche Informationsressource darstellt. Wenn Unklarheit darüber besteht, welcher Stream oder welches Wörterbuch genau den AF-Eintrag enthalten darf, muss das Manifest so nah wie möglich an dem Objekt angehängt werden, das die beschriebene Datenressource tatsächlich speichert.

HINWEIS

Das C2PA-Manifest, das ein Rasterbild beschreibt, würde an den Image XObject-Stream angehängt werden.

, und das Manifest für eingebettete Schriftartdateien würde eher an Schriftartdatei-Streams angehängt werden

als an Schriftartwörterbüchern angehängt.

A.4.4. Beispiel

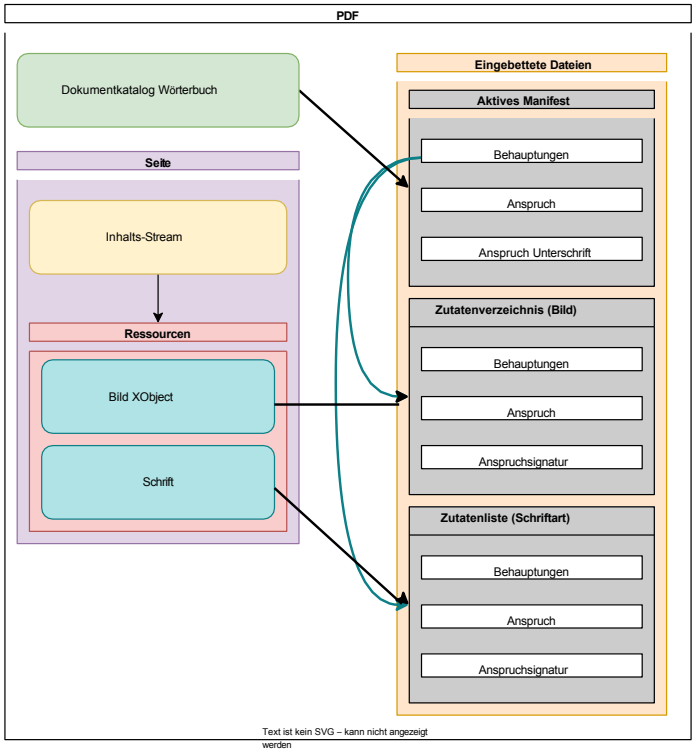

Abbildung 20. Beispiel für eine PDF-Datei mit mehreren Inhaltsverzeichnissen

A.5. Einbetten von Manifesten in BMFF-basierte Assets

A.5.1. Das „uuid”-Feld für C2PA

Alle BMFF-basierten C2PA-Assets, unabhängig davon, ob es sich um zeitgesteuerte (z. B. Videos mit oder ohne Audiospuren), nicht zeitgesteuerte (z. B. Standbilder) oder gemischte (z. B. Live- oder animierte Fotos) audiovisuelle Medien handelt, müssen ein „uuid”-Feld verwenden, das der unten definierten Syntax und Semantik entspricht.

HINWEIS

Der Grund dafür, dass ein „uuid”-Feld anstelle eines „c2pa”-Feldes verwendet wird, ist, dass Browser, die auf Chromium basieren, die Wiedergabe sofort abbrechen, wenn sie auf unbekannte Felder der obersten Ebene stoßen.

Einige BMFF-basierte Dateiformate, die mit dieser Methode unterstützt werden, sind:

- MPEG-4-Codepunkte, entweder vollständig (`.mp4`) oder fragmentiert (`.m4s`); herunterladbare Audiodateien (`.m4a`);
- HEIF (`.heif`, `.heic`);
- AVIF (`.avif`).

A.5.1.1. Definition

```
Box-Typ: „uuid”
```

```
Erweiterter Box-Typ: 0xD8, 0xFE, 0xC3, 0xD6, 0x1B, 0x0E, 0x48, 0x3C, 0x92, 0x97, 0x58, 0x28, 0x87, 0x7E, 0xC4,
0x81
Container: Datei
Obligatorisch: Nein
Anzahl: Null oder mehr
```

Die „uuid”-Box von C2PA bettet die Herkunft in BMFF ein. Eine solche Box enthält einen C2PA-Manifest-Speicher, und es kann eine oder mehrere zusätzliche Boxen geben, die weitere für die Validierung erforderliche Informationen enthalten.

A.5.1.2. Syntax

```
aligned(8) Klasse ContentProvenanceBox erweitert FullBox('uuid', extended_type = 0xD8 0xFE 0xC3 0xD6 0x1B 0x0E
0x48 0x3C 0x92 0x97 0x58 0x28 0x87 0x7E 0xC4 0x81, version = 0, 0) {
     string box_purpose;
     bit(8) data[];
}
```

A.5.1.3. Bezüglich eindeutiger IDs

Es gibt Fälle, wie z. B. fragmentierte MP4-Dateien (fMP4), in denen die ID für einen Teilbereich des Assets, wie z. B. das Feld „track_id” der Box „tkhd”, nur lokal für einen Teilbereich des gesamten Assets eindeutig ist und nicht global für das gesamte Asset.

Da eine global eindeutige ID erforderlich ist, um zu bestimmen, was gehasht werden soll, wird eine eindeutige ID hinzugefügt. Diese eindeutige ID entspricht keinem Wert aus dem ursprünglichen Asset; stattdessen wird jeder Wert bei der Erstellung des Manifests definiert. Die eindeutige ID wird dann mit einer zugehörigen lokalen ID kombiniert, um eine ID zu bilden, die für das gesamte Asset global eindeutig ist.

A.5.2. Semantik

Der Zweck jeder Box (box_purpose) und die davon abhängigen Felder (data) werden im Folgenden für jede Box beschrieben.

A.5.3. Box mit dem Manifest

Das Feld mit dem C2PA Manifest Store muss vor dem ersten „mdat”-Feld in der Datei und vor allen „moov”-Feldern in der Datei erscheinen. Um die Überprüfung von major_brand und compatible_brand zu ermöglichen, muss es nach dem „ftyp”-Feld platziert werden. Wenn das aktive Manifest eines Assets ein Update-Manifest ist, befindet sich der vorherige Standard-C2PA Manifest Store wie oben angegeben, wobei box_purpose in original geändert wurde. Der aktualisierte C2PA Manifest Store muss als letztes Feld der Datei vorhanden sein, wobei box_purpose auf update gesetzt ist.

Die Felder in dem oben beschriebenen entsprechenden Feld sind wie folgt einzustellen.

box_purpose

Für einen C2PA Manifest Store muss dieser Wert manifest, original oder update lauten.

data

Wenn box_purpose „manifest” lautet, müssen die ersten 8 Bytes innerhalb von „data” der absolute Datei-Byte-Offset zur ersten zusätzlichen „uuid”-C2PA-Box mit box_purpose gleich „merkle” sein. Wenn diese Datei keine solchen Boxen enthält, müssen diese 8 Bytes

muss Null sein. Auf diese 8 Bytes folgen die rohen C2PA-Manifest-Speicher-Bytes, gefolgt von null oder mehr ungenutzten Auffüllbytes. Wenn box_purpose „original“ ist, bedeutet dies, dass eine weitere C2PA-Box vorhanden ist, deren box_purpose-Wert auf „update“ gesetzt ist. Die „Daten“ innerhalb dieser ursprünglichen Box bleiben unverändert. Wenn box_purpose „update“ ist, darf der C2PA-Manifest-Speicher nur Aktualisierungsmanifeste enthalten.

HINWEIS

Das Feld „data” innerhalb der Box „uuid” vom Typ manifest oder original enthält die absolute Datei Byte-Offset, Manifest und Auffüllungsbytes. Die Original- und Manifest-Boxen sind bis auf den Wert von box_purpose identisch, sodass die Hash-Bindungen unverändert bleiben. Es werden keine gehashten Daten durch Anhängen einer „update”-Box verschoben.

Füllbytes sind außerhalb des „uuid”-Feldes nicht zulässig, es sei denn, sie sind in einem eigenen mp4-Feld enthalten, z. B. einem „free“-Box enthalten sind.

Bei fragmentierten MP4-Dateien (fMP4) muss in jedem Initialisierungssegment ein identisches „uuid”-C2PA-Feld vom Typ Manifest vorhanden sein; der C2PA-Manifest-Speicher muss identisch sein.

A.5.4. Zusätzliche „c2pa”-Boxen für große und fragmentierte Dateien

A.5.4.1. Allgemeines

Einige Dateien haben eine oder mehrere sehr große „mdat”-Boxen (z. B. große Video- oder Bilddateien, die schrittweise heruntergeladen und gerendert werden können) oder eine große Anzahl unabhängiger „mdat”-Boxen (z. B. fMP4, bei denen jedes Fragment unabhängig heruntergeladen werden kann).

In diesen Fällen ist es unangemessen, von einem Client zu verlangen, alle „mdat”-Boxen vollständig herunterzuladen, bevor ein Teil des Assets validiert werden kann. Diese Notwendigkeit lässt sich durch die Verwendung mehrerer Hashes vermeiden.

Für jede große „mdat“-Box haben Teilmengen der Box individuelle Hashes, die unabhängig voneinander validiert werden können. Wie diese Teilmengen bestimmt werden, wird weiter unten erläutert. Bei fMP4-Inhalten, bei denen jede „mdat“-Box unabhängig voneinander heruntergeladen werden kann, hat jedes Fragment seinen eigenen individuellen Hash.

Im einfachsten Fall werden alle diese Hashes im aktiven Manifest gespeichert. Jede Teilmenge verfügt über eine zusätzliche „uuid”-C2PA-Box, die angibt, wie ihr Hash im aktiven Manifest zu finden ist. Warum dies so ist, erfahren Sie in der obigen Anmerkung zu eindeutigen IDs.

Bei ausreichend großen Assets würde jedoch die Aufnahme aller Hash-Werte der Teilmengen in das Manifest selbst die Größe des C2PA-Manifest-Speichers auf ein oder mehrere Megabyte erhöhen.

Ein so großer C2PA-Manifest-Speicher für große Assets lässt sich durch die Verwendung eines oder mehrerer Merkle-Bäume vermeiden.

- Bei einem großen, nicht fragmentierten Asset, das eine oder mehrere „mdat”-Boxen in einer einzigen großen Datei enthält, wird für jede „mdat”-Box ein Merkle-Baum verwendet.
- Bei einem großen fragmentierten Asset, das eine Reihe von „mdat”-Boxen für einen einzelnen Track enthält, die über mehrere Dateien verteilt sein können, wird für jeden Track ein Merkle-Baum verwendet.

In beiden Fällen gilt:

- Jeder Blattknoten eines beliebigen Merkle-Baums ist der Hashwert der Teilmenge.
- Das Manifest speichert eine Zeile jedes Merkle-Baums.
- Das zusätzliche „uuid”-C2PA-Feld, das für jede Teilmenge vorhanden ist, gibt an, welche Merkle-Baum-Zeile im aktiven Manifest erforderlich ist und welchen Blattknoten es darstellt. Es enthält auch alle zusätzlichen Hash-Werte aus dem Merkle-Baum, die erforderlich sind, um einen Hash-Wert in der Merkle-Baum-Zeile des aktiven Manifests abzuleiten.

Die Auswahl, welche Merkle-Baum-Zeile im Manifest gespeichert werden soll, führt zu einem Kompromiss hinsichtlich der Größe innerhalb des Assets. Insbesondere minimiert die Speicherung eines einzelnen Hash pro Merkle-Baum im Manifest die Größe des Manifests, erfordert jedoch die Speicherung von log2(Teilmengen) in jeder teilmengenspezifischen Box. Jedes Mal, wenn sich die Anzahl der im Manifest für einen Merkle-Baum gespeicherten Hashes verdoppelt (durch Verschieben einer Merkle-Baumzeile „nach unten“), verringert sich die Anzahl der in jeder untermengenbezogenen Box gespeicherten Hashes um eins. Somit verringert eine Vergrößerung des Manifests die Größe des gesamten Assets und umgekehrt, und da Hashes für einzelne Teilmengen über Teilmengen hinweg repliziert werden, um einen im Manifest angegebenen Hash abzuleiten, ist der Kompromiss nicht 1 zu 1.

Die Entscheidung über diesen Kompromiss hinsichtlich der Größe bleibt der Implementierung überlassen, die das Manifest erstellt. Diese Spezifikation schreibt weder vor noch empfiehlt sie, dass eine bestimmte Merkle-Baum-Zeile im Manifest gespeichert wird. Da jedoch der einfachste Fall der Speicherung aller Teilmengen-Hashes im Manifest der Verwendung eines Merkle-Baums entspricht, bei dem die Blattknoten im Manifest gespeichert sind, wird in allen Fällen dieselbe Merkle-Baum-Konstruktion für mehrere Hashes verwendet. Diese Konstruktion ist wie folgt definiert.

Der Teil des Manifests, der den BMFF-Hash enthält, muss das Merkle-Feld enthalten. Weitere Informationen finden Sie in Abschnitt 9.2.3, „Hashing einer BMFF-formatierten Ressource“.

A.5.4.1.1. Nicht fragmentierte Assets, die stückweise validiert werden können

Wenn das Manifest eine Nicht-Blattzeile des Merkle-Baums enthält, müssen zwei oder mehr zusätzliche „uuid”-C2PA-Boxen mit `box_purpose` auf `„merkle”` gesetzt, wie unten beschrieben, in die Datei aufgenommen werden. Sie müssen nicht in die Datei aufgenommen werden, wenn das Manifest die Blattzeile des Merkle-Baums enthält. Wenn sie vorhanden sind, müssen sie auf die letzte „mdat”-Box in der Datei folgen.

Der für einen bestimmten Blattknoten im Merkle-Baum verwendete Hash wird aus der Teilmenge der Nutzlast des `„mdat”` berechnet. Der `„mdat”` wird in Größen unterteilt, die durch `„fixedBlockSize”` oder das Array von `„variableBlockSizes”` in der `bmff-merkle-map` definiert sind, wobei die Summe der `„variableBlockSizes”` der Größe der „mdat”-Nutzlast entsprechen muss.

Alle derartigen zusätzlichen `„uuid”`-C2PA-Boxen müssen die folgenden Anforderungen erfüllen.

- Sie müssen in derselben Reihenfolge wie die Teilmengen vorliegen, die sie hashen, wie durch das Feld `„variableBlockSizes”` angegeben.
- Sie müssen so gruppiert sein, dass die Hilfs-C2PA-Boxen `„uuid”` eines einzelnen Merkle-Baums sequenziell sind und keine dazwischenliegenden Boxen aufweisen.
- Der Positionswert im ersten Feld soll auf 0 gesetzt werden, im zweiten Feld auf 1 und danach

danach sequenziell erhöht werden.

A.5.4.1.2. Fragmentierte Assets

Für fMP4-Assets, die auf mehrere Dateien aufgeteilt sind:

- Ein zusätzliches „uuid"-C2PA-Feld mit `box_purpose` auf `„merkle"` gesetzt, wie unten beschrieben, muss in jeder Fragmentdatei unmittelbar vor dem „moof"-Feld enthalten sein.
- Der für einen bestimmten Blattknoten im Merkle-Baum verwendete Hash muss sich auf alle Daten in der ihn enthaltenden einzelnen Fragmentdatei beziehen, mit Ausnahme der Daten, die durch die Ausschlussliste ausgeschlossen sind.

HINWEIS Diese Spezifikation unterstützt keine fMP4-Assets, die auf mehrere Dateien aufgeteilt sind, wobei einzelne Fragmentdateien mehr als eine „moof"-Box oder „mdat"-Box oder beides enthalten.

Für fMP4-Assets, die als einzelne flache MP4-Datei mit einem einzigen `„moov"` für alle Spuren und dann einem `„moof"` /„mdat"-Paar für jedes Fragment:

- Eine zusätzliche `„uu`id"-C2PA-Box mit `box_purpose` auf `„merkle"` gesetzt, wie unten beschrieben, muss unmittelbar vor jeder „moof"-Box eingefügt werden.
- Der für einen bestimmten Blattknoten im Merkle-Baum verwendete Hash muss über dieser „moof"-Box plus allen Daten vor der nächsten „moof"-Box oder über allen Daten bis zum Ende der Datei liegen, wenn keine weitere „moof"-Box vorhanden ist. Der Hash darf keine Daten abdecken, die durch die Ausschlussliste ausgeschlossen sind.

WICHTIG Wenn Sie eine C2PA-konforme fMP4-Datei nehmen, die auf mehrere Dateien aufgeteilt ist (d. h. „c2pa"-Boxen vom Typ `„manifest"` und `„merkle"` enthält), und die einzelnen Dateien aneinanderhängen, erhalten Sie keine einzige C2PA-konforme Datei (und umgekehrt). Das liegt daran, dass
Boxen in jedem „merkle"-Hash enthalten sind, in den beiden Fällen unterschiedlich sein wird. Wenn beide Formen wünschenswert sind, muss die zweite Form die erste Form als Bestandteil betrachten, und das neue Manifest muss sowohl eine Bestandteil-Assertion mit der Beziehung `parentOf` als auch eine Aktions-Assertion enthalten, die eine Aktion vom Typ `c2pa.repackaged` umfasst.

A.5.4.1.3. Box mit dem Merkle-Hilfsprogramm

Unabhängig davon, wie das Asset strukturiert ist, müssen die Felder in der entsprechenden oben beschriebenen Box wie folgt festgelegt werden.

box_purpose

Für ein zusätzliches „uuid"-C2PA-Feld muss dieser Wert `„merkle"` lauten.

Daten

Wenn `box_purpose „merkle"` ist, muss dieser Wert rohe CBOR-Bytes enthalten, die angeben, wie ein Teil des Assets wie folgt validiert werden soll. Wenn es mehrere zusätzliche „uuid"-C2PA-Boxen mit `box_purpose „merkle"` für einen bestimmten Merkle-Baum in einer einzigen Datei gibt, muss jeder eine ausreichende Anzahl von Füllbytes (null oder mehr) folgen, damit alle zusätzlichen „uuid"-C2PA-Boxen für diesen Merkle-Baum eine feste Größe haben.

HINWEIS Wenn mehr als eine dieser Boxen in einer einzigen Datei vorhanden ist, d. h. wenn es sich um einen Fall handelt, in dem große

Da „mdat" (s) stückweise validiert werden, ist eine feste Größe erforderlich, damit ein Client, der schrittweise herunterlädt, nur die Boxen herunterladen kann, die er für die Validierung benötigt, anstatt den gesamten Merkle-Baum. Ein solcher Client kann anhand des absoluten Dateibyte-Offsets im aktiven Manifest genügend der ersten dieser Boxen herunterladen, um festzustellen, ob seine uniqueId und localId mit dem „mdat" übereinstimmen, das er zu validieren versucht. Ist dies der Fall, kann er den absoluten Dateibyte-Offset zu der Box, die er validieren muss, bestimmen, indem er die Teilmengezahl mit dieser Größe multipliziert und dann nur diese Box herunterlädt. Andernfalls kann er den absoluten Dateibyte-Offset zum Anfang des nächsten Merkle-Baums bestimmen, indem er diese feste Größe mit der aktuellen Gesamtzahl der Blattknoten des Merkle-Baums multipliziert, und er kann diesen Vorgang wiederholen, bis er die benötigte Box gefunden hat. Die Gesamtgröße des Downloads für diese Teilmenge von Boxen ist im Vergleich zur Größe einer einzelnen Teilmenge sehr gering.

A.5.4.2. Schema und Beispiel

Das Schema für diesen Typ wird durch die Regel „bmff-merkle-map" in der folgenden CDDL-Definition definiert:

```
; Die Datenstruktur, die verwendet wird, um ausreichende Informationen zur Validierung einer einzelnen „mdat"-Box
oder

; einen Teil einer „mdat"-Box zu validieren, wenn ein Merkle-Baum
verwendet wird", bmff-merkle-map = {
  „uniqueId": int, ; Eine eindeutige Ganzzahl, die zur Unterscheidung lokaler IDs
  verwendet wird „localId": int, ; Eine lokale ID, die den Merkle-Baum angibt.
  „location": int, ; Nullbasierter Index in die unterste Merkle-Baum-Zeile, die dieser „mdat"-Box oder diesem
Teil dieser „mdat"-Box entspricht
  ? „hashes": [1* bstr], ; Ein geordnetes Array, das die Menge zusätzlicher Hashes darstellt, die
erforderlich sind, um einen Hash im Merkle-Baum zu erreichen, der im Manifest angegeben ist, vom untersten
Blatt (Peer dieses Knotens) bis zum obersten Stamm (Kind des Knotens im Manifest). Beachten Sie, dass dieses
Array möglicherweise nicht vorhanden ist, z. B. wenn das Manifest selbst die unterste Zeile des Merkle-Baums
enthält. Null-Hashes sind in diesem Array nicht enthalten. Der verwendete Algorithmus wird anhand des Feldes
„alg" aus dem entsprechenden Eintrag im Feldarray „merkle" in der BMFF-Hash-Struktur bestimmt.
}
```

Ein Beispiel in CBOR-Diagnoseschrift (RFC 8949, Abschnitt 8) ist unten aufgeführt:

```
{
  „Hashes": [
    b64'TWVub3JhaA=='
  ],
  „localId": 4402,
  „location": 2203,
  „uniqueId": 1339
}
```

Bei nicht fragmentierten Assets muss das Feld „localId" in der bmff-merkle-map das Feld „mdat" angeben. Dabei handelt es sich um einen nullbasierten Index, der die Reihenfolge von „mdat" innerhalb der Datei angibt. Bei fragmentierten Assets muss das Feld „localId" in der bmff-merkle-map auf das Feld „track_id" des Feldes „tkhd" gesetzt werden, das sich auf das zu hashende „mdat" bezieht.

A.5.5. Dynamische Stream-Generierung

Viele Implementierungen von Adaptive Bitrate Streaming (ABR) speichern eine einzige Version eines Assets, z. B. als flache MP4-Datei oder in

ein weiteres Zwischenformat und generieren zum Zeitpunkt der Nutzung individuelle Asset-Streams unter Verwendung verschiedener Codecs, Bitraten usw. Infolgedessen muss ein solcher Server entweder die genannten Streams hashen und jedes Mal, wenn der Inhalt genutzt wird, ein C2PA-Manifest erstellen oder, wenn die Generierung deterministisch ist, die Hashes und C2PA-Manifeste einmalig erstellen und zwischenspeichern und sie dann zum Zeitpunkt der Nutzung einbetten.

A.5.6. Anforderungen an Ausschlusslisten

Für alle `c2pa.hash.bmff.v2`- (veraltet) und `c2pa.hash.bmff.v3`-Assertions müssen die Einträge in Beispiel 18, „Immer ausgeschlossene Boxen", immer in der Ausschlussliste erscheinen. Andere Einträge sind zulässig, aber nicht erforderlich.

Das gesamte C2PA-Feld „`uuid`" muss ausgeschlossen werden. (Das Feld „`data`" stellt sicher, dass andere „uuid"-Felder nicht ausgeschlossen werden.

Beispiel 18. Immer ausgeschlossene Boxen

```
xpath = "/uuid"
data = [ { offset = 8, data = b64'2P7D1hsOSDyS11goh37EgQ==' } ]
```

Die gesamten Boxen „`ftyp`" und „`mfra`" müssen ausgeschlossen werden.

```
xpath = "/ftyp"
```

```
xpath = "/mfra"
```

HINWEIS Frühere Versionen dieser Spezifikation enthielten zusätzliche obligatorische Ausschlüsse, aber es wurde festgestellt, dass deren Ausschluss unsicher ist.

Für alle `c2pa.hash.bmff.v2`- (veraltet) und `c2pa.hash.bmff.v3`-Assertions, bei denen die bmff-hash-map sowohl das Hash-Feld als auch die Merkle-Felder enthält, muss der Eintrag in Beispiel 19, „Zusätzliche immer ausgeschlossene Boxen", in der Ausschlussliste erscheinen.

Beispiel 19. Zusätzliche immer ausgeschlossene Felder

```
xpath = "/mdat"
subset = { { 16, 0 } }
```

HINWEIS Wie in der obigen CDDL-Definition angegeben, schließt die `c2pa.hash.bmff.v3`-Aussage in diesem Fall die gesamte 'mdat'-Box aus, aber es wurde festgestellt, dass dieser Ausschluss unsicher ist.

Wie in der obigen CDDL-Definition angegeben, ist ein relativer Byte-Offset oder ein relativer Byte-Offset plus Länge, der die Länge der Box überschreitet, zulässig; Bytes, die über das Ende der Box hinausgehen, dürfen niemals gehasht werden. Wenn beispielsweise die mdat-Box nur 12 Byte lang ist, wird sie vollständig gehasht, und der oben genannte obligatorische Ausschlusseintrag hat keine Wirkung, obwohl er

weiterhin erforderlich ist.

A.5.7. Zeitgesteuerte Medien-Streams, die weder Audio noch Video sind

Zeitgesteuerte Medien-Streams, die weder Audio- noch Videodaten sind, wie z. B. Text-Streams für Untertitel, die der Anspruchgenerator manipulationssicher machen möchte, werden genauso behandelt wie Audio- und Video-Streams.

A.5.8. Externe Referenzen

Extern referenzierte Inhalte, die innerhalb von BMFF-Boxen deklariert sind, beispielsweise in einer „dref"-, „url"- oder „urn"-Box, die der Anspruchgenerator manipulationssicher machen möchte, dürfen die referenzierende Box **nicht** ausschließen und müssen eine separate Cloud-Datenaussage für jede externe Referenz enthalten, die gehasht werden soll.

A.5.9. Anforderungen an die Größe

Wenn eine BMFF-basierte Ressource 32-Bit-Größen oder Offsets in einem oder mehreren Feldern verwendet, z. B. der „stco"-Box, und das Hinzufügen von Boxen zur Einhaltung dieser Spezifikation die Dateigröße auf über 4 Gigabyte erhöht, ist es Aufgabe des Manifest-Erstellers, die Datei so zu bearbeiten, dass geeignete Größen und Offsets verwendet werden, z. B. durch Ersetzen der „stco"-Box durch eine „co64"-Box, bevor das Manifest erstellt wird.

A.6. Einbetten von Manifesten in ZIP-basierte Formate

A.6.1. Allgemeines

Aufgrund ihrer Langlebigkeit und der Tatsache, dass es sich um offen veröffentlichte Spezifikationen handelt, sind viele Befehlsdateiformate eigentlich ZIP-Archive, jedoch mit einer bestimmten Organisation der Inhaltsdateien. Dazu gehören Formate wie EPUB, Office Open XML, Open Document und OpenXPS.

A.6.2. Hashing

A.6.2.1. Hashing der Dateien

Ein ZIP-basiertes Dateiformat muss mit einem Sammlungsdaten-Hash gehasht werden, wobei jede in der ZIP-Datei enthaltene Datei (mit Ausnahme des C2PA-Manifests selbst) einbezogen werden muss. Der Hash jeder Datei in der Sammlung wird über den `lokalen Dateikopf` der Datei berechnet, gefolgt vom komprimierten und/oder verschlüsselten Inhalt und gegebenenfalls einer Datenbeschreibung. Der verwendete Hash-Algorithmus muss im Feld `„alg"` der Sammlungsdaten-Hash-Struktur angegeben werden.

HINWEIS	Der Grund dafür, dass der Hash über den komprimierten/verschlüsselten Inhalt berechnet wird, besteht darin, eine Validierung ohne die Notwendigkeit, die Datei zu dekomprimieren oder über den Entschlüsselungscode zu verfügen. Dies ist wichtig für Formate, die verschlüsselt werden können, wie beispielsweise EPUB.

A.6.2.2. Hashing des ZIP-Zentralverzeichnisses

Wie in 4.3.12 der ZIP AppNote beschrieben, ist das zentrale Verzeichnis ein Array von zentralen Verzeichnis-Headern – einer pro Datei im ZIP-Archiv. Es wird am Ende des ZIP-Archivs gespeichert und dient dazu, die Dateien im ZIP-Archiv und die erforderlichen

Informationen/Metadaten zu diesen Dateien zu finden. Unmittelbar darauf folgt der End of Central Directory-Datensatz (ZIP AppNote, 4.3.16), der Informationen über das ZIP-Archiv selbst enthält.

Um Manipulationen am ZIP-Zentralverzeichnis zu verhindern, wie z. B. das Hinzufügen neuer Dateien oder das Ändern von Informationen zu vorhandenen Dateien, werden jeder „Zentralverzeichnis-Header" im ZIP-Zentralverzeichnis sowie der „Ende des Zentralverzeichnis-Datensatzes" gehasht. Der Hash wird über den Bytebereich vom ersten Byte des „zentralen Verzeichnis-Headers" bis zum letzten Byte des „Ende des zentralen Verzeichnis-Datensatzes" unter Verwendung des Hash-Algorithmus berechnet, der im Feld `„alg"` der Hash-Struktur der Sammlungsdaten angegeben ist.

HINWEIS Die „zentralen Verzeichnis-Header" werden zusammenhängend gespeichert, unmittelbar gefolgt vom „Ende des zentralen Verzeichnisdatensatzes".

Der resultierende Hashwert wird im Feld `„zip_central_directory_hash"` der Hash-Struktur der Sammlungsdaten gespeichert.

ANMERKUNG Die Verwendung einer speziell benannten Datei in der Dateiliste wurde in Betracht gezogen, aber aufgrund des unten beschriebenen Zwei-Durchlauf-Szenarios nicht akzeptiert.

```
; Ein Array von URIs und den zugehörigen Hashes
$collection-data-hash-map /= { "uris": [1*
  uri-hashed-data-map],
  „alg": tstr .size (1..max-tstr-length), ; Eine Zeichenfolge, die den kryptografischen Hash-Algorithmus
identifiziert, der zur Berechnung des Hash-Werts für jeden Eintrag des Arrays „uris" verwendet wird und aus der
C2PA-Hash-Algorithmus-Identifikatorliste stammt.
    ? „zip_central_directory_hash" : bstr,
}

; Die Datenstruktur, die zum Speichern einer Referenz auf eine URI und deren Hash verwendet wird.
$uri-hashed-data-map /= {
  „uri": relative-url-type, ; relative URI-Referenz „hash": bstr, ;
  Byte-Zeichenkette, die den Hashwert enthält
  ? „size": size-type, ; Anzahl der Datenbytes
  ? „dc:format": format-string, ; IANA-Medientyp der Daten
  ? „data_types": [1* $asset-type-map], ; zusätzliche Informationen zum Datentyp
}

; mit CBOR Kopf (#) und Ende ($) werden in regulären Ausdrücken eingeführt, sodass sie nicht explizit benötigt
werden relative-url-type /= tstr .regexp "[-a-zA-Z0-9@:%._\\+~#=]{2,256}\\.[a-z]{2,6}\\b[-a-zA-Z0-
9@:%_\\+.~#?&//=]*"
```

Da die ZIP-Datei vor Fertigstellung des C2PA-Manifests fertiggestellt sein muss, wird ein Zwei-Durchlauf-Ansatz (wie für JPEG, BMFF und PDF beschrieben) verwendet. Im ersten Durchgang wird eine ZIP-Datei mit einer mit Nullen gefüllten Datei `„content_credential.c2pa"` erstellt und der Hash des ZIP-Zentralverzeichnisses berechnet. Im zweiten Durchgang wird das C2PA-Manifest vervollständigt, einschließlich der Eingabe des Werts für das Feld `„zip_central_directory_hash"`.

Eine mögliche Umsetzung dieses Zwei-Durchlauf-Ansatzes wäre:

- Erstellen einer ZIP-Datei mit einer mit Nullen gefüllten C2PA-Manifest-Speicherdatei (groß genug, um ersetzt werden zu können);
- Berechnen Sie den Hashwert des ZIP-Zentralverzeichnisses.
- Fügen Sie den Hash zum Feld `zip_central_directory_hash` der `collection-data-hash-map` hinzu.

- Vervollständigen Sie das Manifest.
- Überschreiben Sie die mit Nullen gefüllte Datei „content_credential.c2pa“ mit den vollständigen Manifestdaten.

Bei der Erstellung der Datei „content_credential.c2pa“ im ZIP-Archiv muss diese gespeichert (Komprimierungsmethode 0) und darf nicht verschlüsselt werden. Die Felder „General Purpose Bit Flag“ und „crc-32“ müssen auf 0 gesetzt werden. Die Felder für Datum und Uhrzeit können auf den Zeitpunkt der Erstellung des ZIP-Archivs oder auf 0 gesetzt werden. Die Datei kann einen Kommentar enthalten.

A.6.3. Platzierung des Manifest-Speichers

Der C2PA-Manifest-Speicher muss im META-INF-Verzeichnis des ZIP-Archivs mit dem Dateinamen content_credential.c2pa und einem für externe Manifeste empfohlenen Medientyp gespeichert werden. Die Datei muss gespeichert (Komprimierungsmethode 0) und darf nicht verschlüsselt werden.

A.6.4. Digitale Signatur von ZIP-basierten Formaten

A.6.4.1. EPUB

Die digitalen Signaturen von EPUB basieren auf W3C XML DigSig Core, wobei jede signierte Datei im <Manifest>-Element des <Signature>-Elements aufgeführt. Darüber hinaus gibt es keine Unterstützung für die Signierung des ZIP-Zentralverzeichnisses. Daher muss die native Signierung von EPUB vor der Einführung des C2PA-Manifests erfolgen.

A.6.4.2. Office Open XML

Die digitalen Signaturen von OOXML basieren auf W3C XML DigSig Core, wobei jede signierte Datei als <Reference>-Element im <Manifest>-Element des <Signature>-Elements aufgeführt. Darüber hinaus gibt es keine Unterstützung für die Signierung des ZIP-Zentralverzeichnisses. Daher muss die native Signierung von OOXML vor der Einführung des C2PA-Manifests erfolgen.

HINWEIS OpenXPS basiert auf dem gleichen Open Packaging Convention (OPC)-Standard wie OOXML, daher gilt derselbe Ansatz.

Anhang B: Implementierungsdetails für `c2pa.metadata`

Die `c2pa.`metadata-Aussage darf nur die unten beschriebenen Schemata und deren Felder enthalten. Benutzerdefinierte Metadaten-Aussagen können jedoch beliebige Werte aus diesen oder anderen Schemata enthalten.

HINWEIS Eine maschinenlesbare Liste aller gültigen Schemata und ihrer Felder finden Sie auf der C2PA-Spezifikationswebsite.

Die in einer `c2pa.`metadata-Assertion enthaltenen Werte können für die Metadaten-Assertion einzigartig sein oder aus den Standard-„Metadatenblöcken" des Asset-Formats stammen. In beiden Fällen müssen sie gemäß den hier beschriebenen Regeln für die JSON-LD-Serialisierung von XMP serialisiert werden.

B.1. Vollständig unterstützte Schemata

Die folgenden Schemata/Namespaces in Tabelle 15, „Vollständig unterstützte Schemata", werden von allen Signaturnutzern vollständig unterstützt:

Tabelle 15. Vollständig unterstützte Schemata

Name	Namespace
XMP Basic	http://ns.adobe.com/xap/1.0/
XMP-Medienverwaltung	http://ns.adobe.com/xap/1.0/mm/
XMP Paged-Text	http://ns.adobe.com/xap/1.0/t/pg/
Camera Raw	http://ns.adobe.com/camera-raw-settings/1.0/
PDF	http://ns.adobe.com/pdf/1.3/

B.2. Teilweise unterstützte Schemata

Die folgenden Schemata/Namespaces in Tabelle 16, „Teilweise unterstützte Schemata", werden nur teilweise unterstützt.

Tabelle 16. Teilweise unterstützte Schemata

Name	**Namespace**
Dublin Core (DC)	http://purl.org/dc/elements/1.1/
IPTC Core	http://iptc.org/std/Iptc4xmpCore/1.0/xmlns/
IPTC-Erweiterung	http://iptc.org/std/Iptc4xmpExt/2008-02-29/
Exif	http://ns.adobe.com/exif/1.0/
ExifEx	http://cipa.jp/exif/1.0/exifEX
Name	**Namensraum**

Photoshop	http://ns.adobe.com/photoshop/1.0/
TIFF	http://ns.adobe.com/tiff/1.0/
XMP Dynamic Media	http://ns.adobe.com/xmp/1.0/DynamicMedia/
PLUS	http://ns.useplus.org/ldf/xmp/1.0/

B.2.1. Dublin Core (DC)

Es werden nur die folgenden Dublin Core (`dc`)-Eigenschaften unterstützt:

- `dc:coverage`
- `dc:date`
- `dc:format`
- `dc:identifier`
- `dc:language`
- `dc:Beziehung`
- `dc:Typ`

B.2.2. IPTC Core

Es werden nur die folgenden IPTC Core (`Iptc4xmpCore`)-Eigenschaften unterstützt:

- `Iptc4xmpCore:Scene`

HINWEIS Einige IPTC-Core-Eigenschaften wurden durch neuere Versionen im IPTC-Erweiterungsschema ersetzt.

B.2.3. IPTC-Erweiterung

Es werden nur die folgenden IPTC-Erweiterungseigenschaften (`Iptc4xmpExt`) unterstützt:

- `Iptc4xmpExt:DigImageGUID`
- `Iptc4xmpExt:DigitalSourceType`
- `Iptc4xmpExt:EventId`
- `Iptc4xmpExt:Genre`
- `Iptc4xmpExt:ImageRating`
- `Iptc4xmpExt:Bildregion`
- `Iptc4xmpExt:Registrierungs-ID`
- `Iptc4xmpExt:Erstellungsort`

- `Iptc4xmpExt:Angezeigter Ort`
- `Iptc4xmpExt:Maximale verfügbare Höhe`
- `Iptc4xmpExt:MaxAvailWidth`

Für weitere Informationen über diesen, verweisen auf https://www.iptc.org/std/photometadata/specification/IPTC-PhotoMetadata#xmp-namespaces-and-identifiers-2

B.2.4. Exif

Es werden nur die folgenden Exif-Eigenschaften aus Tabelle 17, „Unterstützte Exif-Eigenschaften“, unterstützt:

Tabelle 17. Unterstützte Exif-Eigenschaften

- exif:ApertureValue
- exif:BrightnessValue
- exif:CFAPattern
- exif:Farbraum
- exif:KomprimierteBitsProPixel
- exif:Kontrast
- exif:Benutzerdefinierte Wiedergabe
- exif:Digitalisierungsdatum
- exif:Ursprüngliches Datum und Uhrzeit
- exif:Geräteeinstellungsbeschreibung
- exif:Digitalzoomverhältnis
- exif:ExifVersion
- exif:Belichtungskorrekturwert
- exif:Belichtungsindex
- exif:Belichtungsmodus
- exif:Belichtungsprogramm
- exif:Belichtungszeit
- exif:Dateiquelle
- exif:Blitz
- exif:Blitzenergie
- exif:Blitzbildversion
- exif:Blendenzahl
- exif:Brennweite
- exif:BrennweiteIn35mmFilm
- exif:Auflösung der Fokusebene
- exif:FocalPlaneXResolut ion
- exif:FocalPlaneYResolut ion
- exif:Verstärkungsregelung
- exif:Bild-ID
- exif:ISO-Empfindlichkeit
- exif:Lichtquelle
- exif:Maximalblendenwert
- exif:Messmodus
- exif:OECF
- exif:OffsetZeitOriginal
- exif:PixelXDimension
- exif:PixelYDimension
- exif:VerwandteAudiodatei
- exif:Sättigung
- exif:Aufnahmeszenentyp
- exif:Szenentyp
- exif:Erfassungsmethode
- exif:Schärfe
- exif:Verschlusszeitwert
- exif:RäumlicheFrequenzReaktion
- exif:Spektrale Empfindlichkeit
- exif:Motivbereich
- exif:Motivabstand
- exif:Motumentfernungsbereich
- exif:Motivposition
- exif:Weißabgleich
- exif:GPS-Höhe
- exif:GPS-Höhenreferenz
- exif:GPS-Datumsstempel
- exif:GPS-Zielpeilung
- exif:GPS-ZielpeilungRef
- exif:GPS-Zielentfernung
- exif:GPSDestDistanceRef
- exif:GPSDestLatitude
- exif:GPSDestLongitude
- exif:GPSDifferential
- exif:GPSDOP
- exif:GPSHPositionierungsfehler oder
- exif:GPSImgDirection
- exif:GPSImgDirectionRef
- exif:GPSLatitude
- exif:GPSLongitude
- exif:GPSMapDatum
- exif:GPS-Messmodus
- exif:GPS-Verarbeitungsmethode
- exif:GPSSatelliten
- exif:GPSGeschwindigkeit
- exif:GPS-Geschwindigkeitsreferenz
- exif:GPSStatus
- exif:GPSTimeStamp
- exif:GPS-Track
- exif:GPSTrackRef
- exif:GPS-Versions-ID

B.2.5. ExifEx

Es werden nur die folgenden ExifEx-Eigenschaften unterstützt:

- `exifEX:BodySerialNumber`
- `exifEX:Gamma`
- `exifEX:InteroperabilityIndex`
- `exifEX:ISOSpeed`
- `exifEX:ISOSpeedLatitudeyyy`
- `exifEX:ISO-EmpfindlichkeitBreitengradzzz`
- `exifEX:Objektivhersteller`
- `exifEX:Objektivmodell`
- `exifEX:Objektivseriennummer`
- `exifEX:Objektivspezifikation`
- `exifEX:FotografischeEmpfindlichkeit`
- `exifEX:EmpfohlenerBelichtungsindex`
- `exifEX:Empfindlichkeitstyp`
- `exifEX:Standardausgangsempfindlichkeit`

Weitere Informationen hierzu finden Sie unter https://www.cipa.jp/std/documents/download_e.html?DC-010-2020_E.

B.2.6. Photoshop

Es werden nur die folgenden Photoshop-Eigenschaften unterstützt:

- `photoshop:Kategorie`
- `photoshop:City`
- `photoshop:ColorMode`
- `photoshop:Land`
- `photoshop:Erstellungsdatum`
- `Photoshop:Dokumentvorfahren`
- `Photoshop:Verlauf`
- `Photoshop: ICC-Profil`
- `Photoshop:Bundesstaat`
- `Photoshop:Zusatzkategorien`

- Photoshop:Text-Ebenen
- Photoshop:Übertragungsreferenz
- Photoshop:Dringlichkeit

B.2.7. TIFF

Es werden nur die folgenden TIFF-Eigenschaften unterstützt:

- tiff:BitsProSample
- tiff:Compression
- tiff:DateTime
- tiff:Bildlänge
- tiff:ImageWidth
- tiff:Hersteller
- tiff:Modell
- tiff:Ausrichtung
- tiff:Photometrische Interpretation
- tiff:PlanareKonfiguration
- tiff:Primärfarben
- tiff:Referenzschwarzweiß
- tiff:Auflösungseinheit
- tiff:SamplesPerPixel
- tiff:Software
- tiff:Übertragungsfunktion
- tiff:Weißpunkt
- tiff:X-Auflösung
- tiff:Y-Auflösung
- tiff:YCbCr-Koeffizienten
- tiff:YCbCrPositionierung
- tiff:YCbCr-Unterabtastung

B.2.8. XMP Dynamic Media

Es werden nur die folgenden XMP Dynamic Media (xmpDM)-Eigenschaften aus Tabelle 18, „XMP Dynamic Media-Eigenschaften“, unterstützt:

Tabelle 18. XMP Dynamic Media-Eigenschaften

- xmpDM:absPeakAudioFileP ath
- xmpDM:album
- xmpDM:altTapeName
- xmpDM:altTimecode
- xmpDM:audioChannelType
- xmpDM:Audiokompressor
- xmpDM:Audio-Abtastrate
- xmpDM:Audio-Sampletyp
- xmpDM:BeatSplice-Parameter
- xmpDM:Kamerawinkel
- xmpDM:Kamera-Bezeichnung
- xmpDM:Kameramodell
- xmpDM:Kamerabewegung
- xmpDM:Kommentar
- xmpDM:Beitrag
- xmpDM:Dauer
- xmpDM:Datei-Datenrate
- xmpDM:Genre
- xmpDM:gut
- xmpDM:Instrument
- xmpDM:Intro-Zeit
- xmpDM:Tonart
- xmpDM:Protokollkommentar
- xmpDM:Schleife
- xmpDM:Anzahl der Takte
- xmpDM:Markierungen
- xmpDM:OutCue
- xmpDM:Projektname
- xmpDM:Projektreferenz
- xmpDM:PullDown
- xmpDM:relativePeakAudio Dateipfad
- xmpDM:relativerZeitstempel
- xmpDM:Veröffentlichungsdatum
- xmpDM:Resampling-Parameter
- xmpDM:Skalierungstyp
- xmpDM:Szene
- xmpDM:Aufnahmedatum
- xmpDM:Aufnahmedatum
- xmpDM:Aufnahmeort
- xmpDM:Aufnahmename
- xmpDM:Aufnahmenummer
- xmpDM:Aufnahmegröße
- xmpDM:Lautsprecherplatzierung
- xmpDM:Startzeitcode
- xmpDM:Stretchmodus
- xmpDM:Aufnahmenummer
- xmpDM:Bandname
- xmpDM:Tempo
- xmpDM:Zeitskalenparameter
- xmpDM:Taktart
- xmpDM:Spurnummer
- xmpDM:Tracks
- xmpDM:Video-Alpha-Modus
- xmpDM:Video-Alpha-Premultiplikation ipleColor
- xmpDM:videoAlphaUnityIs Transparent
- xmpDM:Videofarbraum
- xmpDM:Videokompressor
- xmpDM:Video-Halbbildfolge
- xmpDM:videobildfrequenz
- xmpDM:videobildgröße
- xmpDM:Video-Pixel-Seitenverhältnis
- xmpDM:Videopixel-Tiefe
- xmpDM:TeilDerZusammenstellung
- xmpDM:Songtext
- xmpDM:Disc-Nummer

B.2.9. PLUS

Es werden nur die folgenden PLUS-Eigenschaften unterstützt:

- plus:DateinameWieGeliefert
- plus:Erstveröffentlichungsdatum

- `plus:ImageFileFormatAsDelivered`
- `plus:ImageFileSizeAsDelivered`
- `plus:Bildtyp`
- `plus:Version`

Weitere Informationen hierzu finden Sie unter http://ns.useplus.org/LDF/ldf-XMPSpecification.

Anhang C: Überlegungen zur Abkündigung

C.1. Status von Konstrukten

Die folgende Tabelle listet Konstrukte auf, deren Status sich im Zuge der Weiterentwicklung dieser Spezifikation

geändert hat. Es werden die folgenden Statuswerte verwendet:

VERALTET

Construct ist veraltet (Claim-Generatoren dürfen es nicht mehr erzeugen; Validatoren werden gebeten, es zu akzeptieren).

UNDEFINED

Konstrukt ist nicht definiert (Validatoren müssen es ignorieren).

<blank>

Konstrukt wird vollständig unterstützt (Validatoren müssen es akzeptieren).

Tabelle 19. Status von Konstrukten

Konstrukt	Typ	v1.3	v1.4	v2.0	v2.1	v2.2
Zeitstempel-Manifest	Manifest	UNDEFINED	UNDEFINED	UNDEFINED		UNDEFINED
`urn:uuid` Namensraum	Bezeichnung				VERALTET	VERALTET
`urn:c2pa` Namespace	Bezeichnung	UNDEFINED	UNDEFINED	UNDEFINED		
`c2pa.data` (Datenfeld)	Bezeichnung					VERALTET
`c2pa.databoxes` (Datenboxspeicher)	Bezeichnung					VERALTET
`sigTst` timestamp	Zeitstempel				VERALTET	VERALTET
`sigTst2` Zeitstempel	Zeitstempel	UNDEFINED	UNDEFINED	UNDEFINED		
`c2pa.claim`	Assertion			VERALTET	VERALTET	VERALTET
`c2pa.claim.v2`	Assertion	UNDEFINED	UNDEFINED			
`c2pa.actions`	Behauptung					
Konstrukt	**Typ**	**v1.3**	**v1.4**	**v2.0**	**v2.1**	**v2.2**

c2pa.actions.v2	Assertion					
c2pa.asset-Typ	Assertion					VERALTET
c2pa.asset-type.v2	Assertion	UNDEFINED	UNDEFINED	UNDEFINED	UNDEFINED	
c2pa.certificate-status	Assertion	UNDEFINED	UNDEFINED	UNDEFINED	UNDEFINED	
c2pa.embedded-data	Assertion	UNDEFINED	UNDEFINED	UNDEFINED	UNDEFINED	
c2pa.font.info	Assertion	UNDEFINED		VERALTET	VERALTET	VERALTET
c2pa.hash.bmff	Assertion	VERALTET	VERALTET	UNDEFINED	UNDEFINED	UNDEFINED
c2pa.hash.bmff.v2	Assertion				VERALTET	VERALTET
c2pa.hash.bmff.v3	Assertion	UNDEFINED	UNDEFINED	UNDEFINED		
c2pa.hash.collection.data	Assertion	UNDEFINED				
c2pa.hash.Multi-Asset	Assertion	UNDEFINED	UNDEFINED	UNDEFINED	UNDEFINED	
c2pa.ingredient	Assertion			VERALTET	VERALTET	VERALTET
c2pa.ingredient.v2	Behauptung				VERALTET	VERALTET
c2pa.ingredient.v3	Assertion	UNDEFINED	UNDEFINED	UNDEFINED		
stds.metadata	Behauptung	UNDEFINED		VERALTET	VERALTET	VERALTET
c2pa.metadata	Assertion	UNDEFINED	UNDEFINED			
c2pa.thumbnail.claim	Behauptung	UNDEFINED	UNDEFINED	UNDEFINED	UNDEFINED	
c2pa.thumbnail.claim.*	Assertion					VERALTET
c2pa.thumbnail.ingredient	Assertion	UNDEFINED	UNDEFINED	UNDEFINED	UNDEFINED	
c2pa.thumbnail.ingredient.*	Assertion					VERALTET
Konstrukt	**Typ**	**v1.3**	**v1.4**	**v2.0**	**v2.1**	**v2.2**

c2pa.Zeitstempel	Assertion	UNDEFINED	UNDEFINED	UNDEFINED	UNDEFINED	
font.info	Assertion	UNDEFINED	UNDEFINED			
stds.iptc	Assertion		VERALTET	VERALTET	VERALTET	VERALTET
stds.exif	Assertion		VERALTET	VERALTET	VERALTET	VERALTET
stds.schema-org	Assertion		VERALTET	VERALTET	VERALTET	VERALTET
Rolle in Region-Karte	Feld				VERALTET	VERALTET
Akteure in action-items-map-v2	Feld			VERALTET	VERALTET	VERALTET
softwareAgents in actions-map-v2	Feld	UNDEFINED	UNDEFINED	UNDEFINED		
softwareAgentIndex in action-common-map-v2	Feld	UNDEFINED	UNDEFINED			
geändert in action-items-map-v2	Feld				VERALTET	VERALTET
Änderungen in action-items-map-v2	Feld	UNDEFINED	UNDEFINED	UNDEFINED		
instanceID in Parameter-map-v2	Feld				VERALTET	VERALTET
sourceLang Sprache in Parametern-map-v2	Feld	UNDEFINED	UNDEFINED	UNDEFINED		
targetLanguage in parameters-map-v2	Feld	UNDEFINED	UNDEFINED	UNDEFINED		
c2pa.trainedAlgorithmicData	DigitalSourceType					VERALTET
Konstrukt	**Typ**	**v1.3**	**v1.4**	**v2.0**	**v2.1**	**v2.2**

http://c2pa.org/digitalsourcetype/trainedAlgorithmicData	DigitalSourceType	UNDEFINED	UNDEFINED	UNDEFINED	UNDEFINED	
http://c2pa.org/digitalsourcetype/ empty	DigitalSourceType	UNDEFINED	UNDEFINED	UNDEFINED	UNDEFINED	