TECHNICAL SPECIFICATION · DEFENSIVE PUBLICATION

KIShieldCam Media Provenance Specification

Version 1.0

Stand-alone-Einbettung quantensicherer Signaturen und qualifizierter Zeitstempel in Foto- und Videodateien.

Publication Date
2026-04-17

Version
1.0

Status
Published — Open Specification
Addendum: 2026-04-19

Author
KI-Shield UG
(haftungsbeschränkt)

Jurisdiction
Greußen, Thüringen, DE
HRB 524511, AG Jena

Reference Implementation
KIShieldCam iOS App — ursprüngliche Spec-Veröffentlichung mit Build 75 (v1.1.6); aktueller Stand v1.3.11 (Build 92, 19.04.2026). Erweiterungen seit Build 75 sind in **Anhang C** dokumentiert. Öffentlich verfügbar im Apple App Store.

PDF herunterladen | Live-Verifikation testen

Abstract

DEUTSCH

Dieses Dokument spezifiziert ein Verfahren zur direkten Einbettung kryptografischer Beweise in Foto- und Videodateien, so dass die Datei **stand-alone** — ohne Server und ohne externe Dienste — auf Unverfälschtheit und Aufnahmezeitpunkt geprüft werden kann. Die Beweise bestehen aus einer Post-Quantum-Signatur gemäß NIST FIPS 204 (ML-DSA-65), einer klassischen Ed25519-Signatur, einem qualifizierten RFC-3161-Zeitstempel, einer Hash-Chain-Referenz zur vorherigen Aufnahme und einem optionalen Blockchain-Anker. Alle Beweise werden als Adobe-XMP-Metadatenblock in die Mediendatei geschrieben (PNG: iTXt-Chunk; JPEG: APP1-Segment; MP4/MOV: ISO-BMFF uuid-Box mit Adobe-XMP-UUID). Das Verfahren erhält die Abspielbarkeit von Videos (keine Modifikation der stco/co64-Sample-Offsets) und ist unabhängig von bestehenden Verfahren wie C2PA.

ENGLISH

This document specifies a method for directly embedding cryptographic proof data into image and video files, such that the file becomes **stand-alone verifiable** — independent of any server or external service. The proof set consists of a post-quantum signature per NIST FIPS 204 (ML-DSA-65), a classical Ed25519 signature, a qualified RFC 3161 timestamp, a hash-chain reference to the prior capture, and an optional blockchain anchor. All proofs are written as an Adobe XMP metadata block inside the media file (PNG: iTXt chunk; JPEG: APP1 segment; MP4/MOV: ISO-BMFF uuid box bearing the Adobe XMP UUID). The method preserves video playability by leaving stco/co64 sample offsets untouched, and is independent of existing schemes such as C2PA.

ZWECK DER VERÖFFENTLICHUNG

Diese Spezifikation wird als **Defensive Publication** veröffentlicht. Sie dokumentiert die technische Umsetzung von KIShieldCam ab Build 74 (Fotos) bzw. Build 75 (Videos) zum Veröffentlichungsdatum und stellt sie als Stand der Technik öffentlich zur Verfügung. Ziel ist die Gemeinfreiheit der hier beschriebenen Verfahrenskombination und der Ausschluss späterer Patent-Monopole auf die dokumentierten Mechanismen.

1. Motivation

Klassische Bild- und Video-Metadaten (EXIF, XMP ohne kryptografische Signatur) haben keinen Beweiswert: jeder Texteditor kann sie fälschen. Etablierte Verfahren wie C2PA (Coalition for Content Provenance and Authenticity) binden heute klassische Signaturen

(ECDSA, RSA) in Medien ein. Mit dem Aufkommen kryptografisch relevanter Quantencomputer gelten diese Verfahren für Beweise, die Jahrzehnte haltbar sein sollen, als riskant.

Diese Spezifikation ergänzt und ersetzt für ihren Anwendungsbereich klassische Signaturverfahren durch den NIST-standardisierten Post-Quantum-Algorithmus ML-DSA-65 (Dilithium, FIPS 204, August 2024) und kombiniert ihn mit einem qualifizierten RFC-3161-Zeitstempel. Die Kombination liefert einen Beweis, der auch nach Ablösung klassischer Verfahren durch Quantencomputer seinen kryptografischen Wert behält.

2. Terminologie

Die Schlüsselworte `MUSS`, `DARF NICHT`, `SOLLTE`, `KANN` in diesem Dokument sind im Sinne von RFC 2119 zu verstehen.

Aufnahme: eine Foto- oder Videodatei, die durch eine konforme Implementierung erzeugt wurde.
Raw-Hash: SHA-256 der Aufnahme *vor* Einbettung der hier beschriebenen Metadaten.
Konforme Implementierung: ein Programm, das diese Spezifikation beim Erstellen oder Verifizieren einer Aufnahme einhält.

3. Metadaten-Schema

Alle Metadaten liegen im XMP-Namensraum `https://ki-shield.de/shieldcam/1.0/` mit dem empfohlenen Präfix `shieldcam`. Konforme Implementierungen **MÜSSEN** die Pflichtfelder setzen und **KÖNNEN** die optionalen Felder setzen.

Feld	Typ	Pflicht	Beschreibung
`id`	UUID	MUSS	Eindeutige Kennung der Aufnahme.
`captureDate`	ISO 8601	MUSS	Client-Zeitstempel (Aufnahmegerät).
`imageHash`	hex (64)	MUSS	SHA-256 der rohen Aufnahme (ohne XMP).

Feld	Typ	Pflicht	Beschreibung
`mediaType`	string	MUSS	photo oder video.
`ed25519Signature`	base64	MUSS	Ed25519 über imageHash.
`ed25519PublicKey`	base64	MUSS	Ed25519 Public Key (Aufnahmegerät).
`chainIndex`	integer	MUSS	Position in der geräte-lokalen Hash-Chain.
`previousHash`	hex (64)	SOLL	Hash der vorherigen Aufnahme.
`rfc3161TimestampToken`	base64 DER	SOLL	RFC 3161 TimeStampResp einer qualifizierten TSA.
`rfc3161TimestampIsLocal`	bool	SOLL	True falls nur lokaler Fallback.
`pqAlgorithm`	string	SOLL	Algorithmen-ID, hier ML-DSA-65.
`pqSignatureHex`	hex	SOLL	ML-DSA-65 Signatur über pqSignedData.
`pqPublicKeyHex`	hex	SOLL	ML-DSA-65 Public Key des Signatur-Dienstes.
`pqSignedData`	string	SOLL	Kanonisch normalisierter Signatur-Input (§5.3).
`txHash`	hex	KANN	Blockchain-Anker-Transaktion.
`txStatus`	string	KANN	pending / confirmed.
`polygonscanURL`	URL	KANN	Blockchain-Explorer-Link.
`deviceModel / deviceOS / appVersion / appBuild`	string	SOLL	Provenance der Aufnahme-Umgebung.
`videoDuration / videoResolution`	float / string	KANN	Nur bei mediaType=video.

4. Einbettung in die Mediendatei

4.1 PNG — iTXt-Chunk

Bei PNG-Dateien wird ein `iTXt`-Chunk (ISO/IEC 15948-1) vor dem IEND-Chunk eingefügt. Das Keyword lautet `XML:com.adobe.xmp`. Compression-Flag **MUSS** 0 sein (unkomprimiert). Language-Tag und Translated-Keyword sind leer. Es folgt der XMP-Payload als UTF-8. Der CRC-32 wird über Type + Data berechnet (Polynom 0xEDB88320, Standard-PNG-CRC).

```
[length 4 BE] [type 'iTXt'] [keyword 'XML:com.adobe.xmp' \0]
[cflag \0] [cmethod \0] [lang \0] [tkey \0] [xmp UTF-8] [crc32 4 BE]
```

4.2 JPEG — APP1-Segment

Bei JPEG wird der XMP-Block als APP1-Segment gemäß Adobe XMP Specification eingebettet. Signatur-Präfix `http://ns.adobe.com/xap/1.0/\0` gefolgt vom UTF-8-XMP-Payload. Konforme Implementierungen **MÜSSEN** das Segment vor dem SOS-Marker platzieren.

4.3 MP4/MOV — ISO-BMFF uuid-Atom

Bei Videos im ISO Base Media File Format (MP4, MOV) wird ein `uuid`-Atom gemäß ISO/IEC 14496-12 am **Ende der Datei** angehängt. Die UUID lautet `BE7ACFCB-97A9-42E8-9C71-999491E3AFAC` (Adobe XMP UUID).

```
[size 4 BE, = 8 + 16 + len(xmp)] [type 'uuid'] [uuid 16 bytes] [xmp UTF-8]
```

Integritätsbedingung: Das Atom wird **ausschliesslich am Ende** der Datei angehängt (append-only). Dadurch bleiben alle Sample-Offsets in `stco` und `co64` des vorhandenen `moov`-Box gültig. Das Video bleibt in QuickTime, iOS Photos, VLC und sämtlichen spezifikationskonformen Playern voll abspielbar.

5. Hash-Bindung und Ketten-Integrität

5.1 Raw-Hash-Definition

Der `imageHash` ist der SHA-256 der Aufnahme **vor** Einbettung der shieldcam-Metadaten. Alle Signaturen beziehen sich auf diesen Hash. Damit bleibt die Signatur stabil, auch wenn später weitere (nicht-shieldcam) Metadaten angefügt werden. Verifier, die den Hash gegen die Datei nachrechnen wollen, **MÜSSEN** die shieldcam-Metadaten vorher entfernen.

5.2 Hash-Chain

Jede Aufnahme verweist über `previousHash` und `chainIndex` auf die vorherige Aufnahme desselben Geräts. Dies verhindert, dass einzelne Aufnahmen nachträglich aus einem Beweis-Korpus entfernt oder eingefügt werden, ohne die Kette zu zerstören.

5.3 Kanonische Form für pqSignedData

Der PQ-Signatur-Dienst signiert einen kanonisch normalisierten String, bestehend aus Feldern mit Pipe-Trennung:

```
pqSignedData := imageHash | chainHash | chainIndex | serverTimestamp | saltOrNonce

Beispiel:
6907d05d79c2f6b3e7346bb10098f49c33cebbd1ab3d23f98e5f50dff10da295|
a692a4afe5f259752e484d6b564a5e045d14e826c0fdb7c12c5a00fa4da8088e|
125|
2026-04-16T19:40:05.354363+00:00|
fa8b9cea4d53759a6ce9893edb54089cb37d1313ddf36930e4f73344680b8b65
```

6. Verifikationsverfahren

Ein Verifier **MUSS** die folgenden Schritte in der angegebenen Reihenfolge ausführen:

1. XMP-Block aus der Mediendatei extrahieren (§4).
2. Alle shieldcam-Pflichtfelder parsen (§3). Fehlt ein Pflichtfeld, verweigert der Verifier.
3. ML-DSA-65-Signatur (pqSignatureHex) gegen pqSignedData und pqPublicKeyHex prüfen (NIST FIPS 204).
4. RFC-3161-Token gegen Signer-CA und Root-CA prüfen. Der Digest im Token **MUSS** gleich imageHash sein.
5. Ed25519-Signatur über imageHash gegen ed25519PublicKey prüfen.

6. Falls Blockchain-Anker vorhanden: Transaktion auf spezifizierter Chain abrufen und imageHash bestätigen.
7. Gesamturteil *fully_valid* nur wenn alle obligatorischen Schritte bestanden werden.

7. Sicherheitsbetrachtungen

Die Spezifikation schützt die Unverfälschtheit einer Aufnahme ab dem Zeitpunkt ihrer Erzeugung auf einem konformen Gerät. Sie schützt **nicht** gegen Manipulation *vor* Signatur. Der Schutz der Signaturschlüssel ist Aufgabe der Implementierung; Referenzimplementierungen **SOLLTEN** hardwaregestützte Key-Speicher (Secure Enclave, StrongBox, TPM) verwenden.

8. Datenschutzbetrachtungen

Die Spezifikation schreibt keine GPS- oder Standortdaten vor und empfiehlt explizit, Ortsdaten **nicht** in das Metadatenfeld-Set aufzunehmen. Verifier **SOLLTEN** Felder mit dem Präfix `gps*` oder Namen wie `latitude`, `longitude`, `altitude`, `location` in der Anzeige ausblenden.

9. Referenzen

`NIST FIPS 204` — Module-Lattice-Based Digital Signature Standard (ML-DSA). NIST, 2024.

`RFC 3161` — Internet X.509 PKI Time-Stamp Protocol. IETF, 2001.

`RFC 2119` — Key words for use in RFCs to Indicate Requirement Levels. IETF, 1997.

`RFC 8032` — Edwards-Curve Digital Signature Algorithm (EdDSA). IETF, 2017.

`ISO/IEC 15948-1` — PNG Specification.

`ISO/IEC 14496-12` — ISO Base Media File Format (MP4 / MOV).

`Adobe XMP` — Extensible Metadata Platform Specification.

`FIPS 180-4` — Secure Hash Standard (SHS): SHA-256.

`C2PA 2.0` — Coalition for Content Provenance and Authenticity, Technical Specification.

Anhang A: Beispiel-XMP-Paket (Foto)

```
<?xpacket begin="" id="W5M0MpCehiHzreSzNTczkc9d"?>
<x:xmpmeta xmlns:x="adobe:ns:meta/" x:xmptk="KIShieldCam">
 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description rdf:about=""
    xmlns:shieldcam="https://ki-shield.de/shieldcam/1.0/"
   shieldcam:id="1FEA0DB1-15C7-4F89-8277-36A7541EDB03"
   shieldcam:captureDate="2026-04-16T19:40:05.423Z"
   shieldcam:imageHash="6907d05d79c2f6b3e7346bb10098f49c33cebbd1ab3d23f98e5f50dff10da295
   shieldcam:mediaType="photo"
   shieldcam:ed25519Signature="zSr7x6ooEnOKsEs9NbANMMU10JLVgxY+F..."
   shieldcam:ed25519PublicKey="Ew/SwY/jiNByONa5i3udVVAJMTwQMgbl..."
   shieldcam:rfc3161TimestampToken="MIIQOzADAgEAMIIQMgYJKoZIhvcN..."
   shieldcam:chainIndex="125"
   shieldcam:previousHash="35b1a84d9ddf7745f09b30ae13a6d8fe9e6b2..."
   shieldcam:pqAlgorithm="ML-DSA-65"
   shieldcam:pqSignatureHex="15dc4a880f7f8f4a82380a994304c18cb1..."
   shieldcam:pqPublicKeyHex="7b40a18ecfe984d1fbed468eba63a248d9..."
   shieldcam:pqSignedData="6907d05d...|a692a4af...|125|2026-04-16T...|fa8b9cea..."
   shieldcam:txHash="f243048afbb0006dbe5c407d927fb72b6cf818fc0b9..."
   shieldcam:txStatus="confirmed"/>
 </rdf:RDF>
</x:xmpmeta>
<?xpacket end="w"?>
```

Anhang B: Prior-Art-Statement

Die in diesem Dokument beschriebenen Verfahren, Algorithmenkombinationen und Datenstrukturen werden durch die Veröffentlichung dieses Dokuments am **17. April 2026** auf `https://kishieldcam.de/whitepaper` zum Stand der Technik. Die beschriebene Umsetzung ist in der KIShieldCam iOS-Anwendung ab Version 1.1.6 (Build 75) implementiert und seit dem 16. April 2026 im Apple App Store öffentlich verfügbar.

Ursprüngliche Veröffentlichung (17. April 2026): Creative Commons Attribution 4.0 International Lizenz (CC BY 4.0). **Ab 19. April 2026** umgestellt auf Creative Commons Attribution-NonCommercial 4.0 International Lizenz (**CC BY-NC 4.0**). Nicht-kommerzielle Parteien (Forschung, Lehre, Privatpersonen, journalistische Arbeit, nicht-gewinnorientierte Organisationen) sind weiterhin berechtigt, die beschriebenen Verfahren zu implementieren,

zu verändern und zu verbreiten, sofern der Urheber (KI-Shield UG, Greußen) genannt wird. **Für kommerzielle Implementierungen, Produkte oder Derivate ist eine ausdrückliche schriftliche Genehmigung der KI-Shield UG erforderlich; Lizenzanfragen an info@ki-shield.de.** Kopien, die vor dem 19. April 2026 unter CC BY 4.0 bezogen wurden, bleiben für ihren jeweiligen Bestand unter dieser ursprünglichen Lizenz (CC-Lizenzen sind nicht widerruflich für bereits erteilte Rechte).

Hinweis zu Schutzrechten: unabhängig von dieser Defensive Publication bestehen beim Deutschen Patent- und Markenamt eingetragene bzw. angemeldete Schutzrechte zu KIShieldCam (4 Anmeldungen, davon 2 eingetragen, Stand April 2026). Diese betreffen abgrenzbare andere Aspekte der Anwendung und werden durch diese Spezifikation weder aufgehoben noch eingeschränkt.

Anhang C: Implementation Update (Addendum, Stand 2026-04-19)

Dieser Anhang dokumentiert die seit der Erstveröffentlichung (Build 75, 17.04.2026) in der Referenzimplementierung umgesetzten Erweiterungen. Die Spezifikation Version 1.0 wird dadurch **nicht** modifiziert; die hier beschriebenen Funktionen sind streng additive, optionale Erweiterungen. Der Konformitätsnachweis folgt in Abschnitt C.7.

C.1 On-Device Post-Quantum-Schlüssel (ab Build 81)

Der ML-DSA-65-Signaturschlüssel wird in der Referenzimplementierung im **Apple Secure Enclave** erzeugt und gespeichert. Der Private Key verlässt das Gerät nie. Die verschlüsselte Schlüsselrepräsentation liegt zusätzlich in der iOS Keychain (Accessibility: `WhenUnlockedThisDeviceOnly`), um den Wiederanlauf nach App-Neustart zu ermöglichen. Voraussetzung: iOS 26 oder höher.

C.2 KiShield-eigene RFC-3161-TSA

Der RFC-3161-Endpunkt der Referenzimplementierung wird unter `https://ki-shield.de/shieldcam/api/v1/timestamp` betrieben. Die zugrundeliegende Systemzeit wird durch `chrony` gegen vier unabhängige NTP-Quellen synchronisiert (PTB Braunschweig, Cloudflare NTS, Hetzner, pool.ntp.org). Der gemessene Synchronisations-Offset gegen

Cloudflare NTS liegt typischerweise bei **~21 Mikrosekunden** (Stand 2026-04-19); die System-Zeit-Abweichung zur NTP-Referenz ist sub-Mikrosekunde. Bei Server-Nichterreichbarkeit fällt die App auf einen lokalen Token-Generator zurück und markiert dies im XMP über `rfc3161TimestampIsLocal=true`.

C.3 Apple App Attest (ab Build 76)

Die Referenzimplementierung integriert Apple App Attest (`DCAppAttestService`). Pro Aufnahme wird eine Assertion über `imageHash | chainHash | chainIndex` erzeugt und als Feld `shieldcam:appleAppAttest` in die XMP-Metadaten geschrieben. Die Assertion belegt, dass die Aufnahme von einer integren App-Instanz auf einem nicht-jailbroken Gerät stammt. Die dafür genutzten Attest-Schlüssel werden von Apple im Secure Enclave generiert und sind an die App-Bundle-ID gebunden.

C.4 Blockchain-Anker-Retry-Queue (ab Build 82)

Schlägt der initiale Anker-Upload fehl, wird die Aufnahme in eine persistente Retry-Queue (`shieldcam-anchor-retry-queue.json` im Dokumente-Verzeichnis) eingetragen. Die Queue arbeitet im Hintergrund mit exponentiellem Backoff (30s, 120s, 600s, 1800s, 7200s, 21600s, 86400s) und maximal 20 Versuchen pro Eintrag. Erfolgreiche Anker werden im XMP nicht nachträglich geändert; der Anker-Status wird im lokalen Audit-Log (siehe C.5) und in der Server-Datenbank geführt.

C.5 Tamper-Evident Audit-Log (ab Build 88)

Die Referenzimplementierung schreibt einen lokalen Audit-Log (`shield-cam-audit.json`) mit SHA-256-Hash-Verkettung (`prevLogHash → entryHash`). Erfasste Ereignistypen umfassen u. a.: `capturePhoto`, `captureVideo`, `keyCreateEd25519`, `keyCreatePQ`, `keyRestoreEd25519`, `keyRestorePQ`, `anchorSuccess`, `anchorFailedEnqueued`, `retrySuccess`, `retryScheduled`, `retryExhausted`, `chainGenesis`, `chainEpochReset`, `appAttestSuccess`, `appAttestFailed`, `logExported`. Die Log-Datei verbleibt auf dem Gerät und kann vom Nutzer signiert exportiert werden.

C.6 Erweiterte XMP-Felder (additiv zu §3)

Feld	Typ	Pflicht	Beschreibung
`deviceOS`	string	SOLL	iOS-Version des Aufnahmegeräts.
`appVersion`	string	SOLL	App-Version (z.B. 1.3.11).
`appBuild`	string	SOLL	App-Buildnummer (z.B. 92).
`appleAppAttest`	base64	KANN	Apple App Attest Assertion (siehe C.3).

C.7 Zero-Knowledge-Eigenschaft der Server-Kommunikation

Die Server-seitigen Endpunkte (TSA, Blockchain-Anker) sehen **niemals**: die Original-Mediendatei, Private Keys (Ed25519, ML-DSA-65), iCloud-Keychain-Secrets. Übertragen werden ausschließlich: `imageHash` (SHA-256), Signaturen und Public Keys, Chain-Metadaten (`chainIndex`, `chainHash`, `previousHash`), der RFC-3161-Token sowie – falls aktiviert – die App-Attest-Assertion. Die Authentizität der Aufnahme ist damit durch Public-Key-Kryptografie belegbar, ohne dass der Server die Rohdaten erhält.

C.8 Share-sichere Verteilung via UTI-Override (ab Build 88)

Messenger wie WhatsApp, Telegram oder Signal erkennen den *Uniform Type Identifier* (UTI) einer geteilten Datei automatisch — `.mp4` wird zu `public.movie`, `.png` zu `public.image`. Die Messenger leiten die Datei daraufhin in ihren **Compression-Pathway** ("als Foto / als Video senden") und re-encodieren die Bytes. Dadurch wird der SHA-256-Hash invalidiert und die Ed25519-/ML-DSA-65-Signaturen werden ungültig.

Die Referenzimplementierung umgeht dieses Verhalten, indem sie beim Teilen via `UIActivityViewController` einen eigenen `UIActivityItemSource`-Adapter (`ByteIdenticalShareItem`) verwendet, der den UTI explizit auf `public.data` setzt:

```
func activityViewController(_ activityViewController: UIActivityViewController,
                            dataTypeIdentifierForActivityType activityType: UIActivity.A
    UTType.data.identifier  // "public.data"
}
```

Die Zielanwendung interpretiert die Datei dadurch als generisches Dokument und verwendet ihren **Document-Pathway** (Datei wird byte-identisch, ohne Re-Encoding, als Anhang

übertragen). Empfänger können die Datei danach aus dem Messenger herunterladen und offline mit einem Verifier nach §6 der Spezifikation prüfen — die kryptografische Kette Hash → Signatur → TSA → Anchor bleibt unverändert intakt.

Praktische Konsequenz: Eine mit KIShieldCam aufgenommene Datei bleibt beim Teilen via WhatsApp, Telegram, Signal, Mail-Anhang oder AirDrop vollständig verifizierbar, sofern der Empfänger sie als Datei (nicht als neu eingebettetes Medium) öffnet.

C.9 Default-Speicherpfade — Fotos & Videos

Die Referenzimplementierung schreibt jede Aufnahme primär in das App-Sandbox-Verzeichnis `Documents/KiShieldCam-Aufnahmen/`. Dieser Ordner ist über die **Apple Dateien-App** (Files.app) sichtbar; damit liegt die Datei an einem Ort, an dem sie byte-identisch gelesen und weitergegeben werden kann. Speicherkommando:

```
// Foto: PNG inkl. eingebettetem XMP-Block
try EvidenceMediaStore.shared.savePhoto(imageData, for: evidence.id)

// Video: byte-identische Kopie aus dem Capture-URL, kein Re-Encoding
try EvidenceMediaStore.shared.saveVideo(from: sourceURL, for: evidence.id)
```

Zusätzlich bietet die App einen optionalen Auto-Export in die **Apple Fotos-App** (`PHPhotoLibrary`, Authorization `.addOnly`): Fotos werden dabei als `PHAssetCreationRequest.addResource(.photo, data:)` hinzugefügt. Videos werden zwar ebenfalls in die Fotos-App exportiert, doch der **empfohlene Share-Pfad** bleibt die Dateien-App — dort bleibt die Datei byte-identisch. Die Fotos-App kann bei Videos ein container-weites Re-Encoding durchführen, wodurch der SHA-256-Hash invalidiert würde.

Praktische Empfehlung für forensische Nutzung: Originaldatei stets aus der Dateien-App (`Auf meinem iPhone → KiShieldCam-Aufnahmen`) teilen, nicht aus der Fotos-App. In Kombination mit dem UTI-Override aus C.8 bleibt die komplette Beweis-Kette (Hash → Signatur → TSA → Anker) über Messenger, Mail und AirDrop stabil.

C.10 Konformitätsnachweis

Alle in C.1–C.9 beschriebenen Mechanismen sind streng additiv: Sie erweitern die Spezifikation v1.0 ohne ihre Pflichtfelder, Verifikationsschritte oder Schemastruktur zu ändern. Jeder Verifier, der nach §6 von Spec v1.0 arbeitet, verifiziert eine Aufnahme der

Referenzimplementierung v1.3.11 erfolgreich, sofern alle Spec-Pflichtfelder gesetzt sind. Die Erweiterungen erhöhen die Beweiskraft und Praxistauglichkeit, sind aber zur Erfüllung der Spezifikation nicht erforderlich.

Ende der Spezifikation Version 1.0

KI-Shield UG (haftungsbeschränkt) · HRB 524511 Amtsgericht Jena · Greußen, Thüringen
Veröffentlicht am 17. April 2026 · Lizenz seit 19. April 2026: CC BY-NC 4.0 · Kommerzielle Lizenz: info@ki-shield.de