DAVID KORAN & ASSOCIATES
CMMC Compliance Consulting

LEARNING DISCOVERY DOCUMENT

File Hashing for CMMC Artifact Integrity

Proving Evidence Authenticity to Your C3PAO

Prepared for:
IT Administrators and Compliance Officers
Defense Industrial Base (DIB) Contractors

March 2026

Section 1: Executive Summary

Integrity is the foundational assurance that the evidence you present during a CMMC readiness review accurately reflects the state of your environment at the time it was collected, and that it has remained unaltered since.

The CIA triad, comprising Confidentiality, Integrity, and Availability, sits at the heart of NIST SP 800-171 and CMMC 2.0. While confidentiality protections for Controlled

Unclassified Information (CUI) often consume the most attention during readiness preparation, integrity is the pillar that verifies that all of the other controls are working as documented. Without it, no artifact can be trusted.

File hashing is the technical mechanism that enforces integrity for documentation artifacts. A cryptographic hash function takes any input file and produces a fixed-length string of characters called a digest. If even a single bit of the file changes after the hash was generated, the resulting digest is completely different. This property makes hashes an unforgeable seal on evidence.

Within a CMMC context, the practical importance of this cannot be overstated. Third-Party Assessment Organizations (C3PAOs) are charged with verifying that your security practices are *real, current, and accurate*. A policy document, a screenshot, a system-generated log, or a configuration export that has been altered after collection, whether accidentally or intentionally, is worthless as evidence. A properly documented hash manifest transforms a folder of files into a verifiable, tamper-evident evidence package.

This Learning Discovery Document gives your IT administrators and compliance officers a complete technical and procedural foundation for implementing file hashing as a mandatory step in your evidence collection and management workflow.

Pillar	Role in CMMC Evidence Management
Confidentiality	Protecting CUI during storage and transmission of evidence artifacts.
Integrity	Cryptographically verifying that evidence files have not been modified since collection. This is the primary subject of this document.
Availability	Ensuring that evidence repositories and hash manifests are accessible to authorized reviewers throughout the review period.

Section 2: Technical Standards

Not all hash algorithms are created equal, and in a DoD-related compliance environment, the choice of algorithm is a matter of policy, not preference.

> **A NOTE FROM THE AUTHOR**
>
> *My introduction to cryptographic integrity came in 1995, through Phil Zimmermann's PGP and the early SHA-1 implementations just beginning to circulate in the security community. That exposure moved quickly from theory to practice. In the mid to late 1990s, while designing and developing banking and brokerage server systems, I worked with both MD5 and SHA-1 in production environments. MD5 was widely adopted at the time as a way to eliminate plaintext password storage in web application databases; rather than storing credentials directly, we stored the MD5 digest and compared hashes during authentication. It felt like a significant improvement over what had come before, and for its era, it was. I also built our own file integrity mechanisms using SHA-1 to detect tampering by external actors targeting financial infrastructure. When we undertook rewrites of core banking software during that same period, maintaining the integrity of the codebase itself was a genuine operational concern: we hashed source files at known-good states and compared them against working copies at key checkpoints throughout the rewrite. It was painstaking work by today's standards, but it built a discipline around verification that has carried forward into every compliance engagement I've worked on since. The algorithms I relied on then are now prohibited **for cryptographic integrity in federal systems**; that evolution is precisely why algorithm selection is covered in detail in this document.*

2.1 A Brief History of Cryptographic Hash Algorithms

Three algorithms dominate the conversation for compliance professionals: MD5, SHA-1, and SHA-256. Each generation replaced the previous due to the discovery of practical collision vulnerabilities: the ability for an attacker to produce two different files with the same hash digest.

Algorithm	Digest Length	Status	DoD Acceptable?	Notes
MD5	128-bit (32 hex chars)	**Cryptographically Broken**	No	Practical collisions demonstrated since 2004.

Algorithm	Digest Length	Status	DoD Acceptable?	Notes
				Retained only for non-security checksums.
SHA-1	160-bit (40 hex chars)	**Deprecated**	No	NIST deprecated in 2011. SHAttered collision attack published 2017. Prohibited for new use.
SHA-256	256-bit (64 hex chars)	**Approved / Current**	**Yes**	FIPS 180-4 approved. NSA Suite B. Required by NIST SP 800-131A Rev. 2 for integrity uses.
SHA-3 / SHA-512	256-bit / 512-bit	Approved	Yes	Acceptable alternatives. SHA-256 is most widely supported in enterprise tooling.

2.2 Why MD5 and SHA-1 Are Insufficient for DoD-Related Compliance

The prohibition on MD5 and SHA-1 in DoD-adjacent compliance isn't a theoretical concern. It derives from documented policy and demonstrated cryptanalytic attacks.

- **MD5 Collision Weakness:** In 2004, researchers demonstrated that two different files can be engineered to produce the same MD5 digest. By 2008, forged SSL certificates using MD5 collisions were demonstrated in practice. Using MD5 as an integrity mechanism means a sophisticated actor could modify

an artifact and produce a matching hash, thereby defeating your entire evidence-integrity posture.

- **SHA-1 SHAttered Attack:** In 2017, a team from Google and CWI Amsterdam produced the first publicly known SHA-1 collision, requiring only 110 GPU-years of compute. What once required nation-state resources is now within reach of well-resourced threat actors. NIST SP 800-131A Rev. 2 prohibits SHA-1 for generating digital signatures and hash-only applications in new systems.
- **CMMC and FIPS Alignment:** CMMC 2.0 requires that cryptographic protections align with FIPS-validated modules where applicable. FIPS 140-2 and its successor FIPS 140-3 approve the SHA-2 family (including SHA-256) for integrity verification. MD5 and SHA-1 are not listed as approved algorithms for this purpose.
- **C3PAO Expectations:** A C3PAO reviewer encountering an MD5-based hash manifest will flag it as a deficiency. The algorithm itself serves as evidence of inadequate cryptographic hygiene, potentially triggering additional scrutiny of your entire control implementation.

2.3 SHA-256 in Practice: What the Digest Looks Like

Understanding the output format helps your team recognize and verify hashes correctly. A SHA-256 digest is always exactly 64 hexadecimal characters, regardless of the size of the input file.

Input	SHA-256 Digest
Empty file (0 bytes)	`e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855`
"Hello, World!"	`dffd6021bb2bd5b0af676290809ec3a53191dd81c7f70a4b28688a362182986d`
Same file, 1 bit changed	*Completely different 64-character string (avalanche effect)*

This avalanche effect is the core integrity property. There is no partial match; even the smallest modification produces an entirely unrecognizable digest.

> **A NOTE ON SALTING**
>
> *A common question from technically experienced readers is whether SHA-256 requires a salt for CMMC artifact integrity purposes. The answer is no, and the reason matters.*

> *Salting is a technique used in credential storage: a random value is added to a password before hashing so that two identical passwords produce different digests, defeating precomputed rainbow table attacks. That problem does not exist in artifact integrity verification. Here, the goal is the opposite: the digest must be fully deterministic and reproducible so that any party with the same file and the same algorithm arrives at exactly the same result. Introducing a salt would break the verification chain entirely. SHA-256 without salting is correct and complete for this use case. If your organization is also using SHA-256 for password or credential storage, that is a separate control domain governed by NIST SP 800-132, and purpose-built key derivation functions such as PBKDF2, bcrypt, or Argon2 are the appropriate mechanisms there, not bare SHA-256.*

Section 3: The Lifecycle of an Artifact

Every piece of evidence you submit for review has a lifecycle: it's created, stored, and presented. Integrity controls must be applied at each phase.

3.1 Phase 1: Point of Collection (Generating the Initial Hash)

The hash must be generated at the moment of artifact creation or collection, before any other handling occurs. Generating the hash after the file has been moved, renamed, or opened by another application introduces unnecessary risk and a potential chain-of-custody challenge.

The governing principle: Hash first, then file. Never file first, then hash.

The following procedures apply to the three most common evidence collection scenarios:

- **System-generated exports (logs, configuration files):** Run the hash command in the same terminal session or script that exports the file. Document the timestamp of both the export and the hash generation.
- **Screenshots and screen recordings:** Hash immediately after saving the file to disk. Don't open the file in an image editor or any application before hashing.

- **Third-party reports (vulnerability scanner outputs, GRC exports):** Hash the downloaded file before uploading it to your evidence repository. Many GRC platforms offer built-in hashing, but you should verify it independently.

OS / Platform	Hash Command	Output Format
Windows (PowerShell)	`Get-FileHash .\file.pdf -Algorithm SHA256`	Object with Hash property (64-char hex)
macOS / Linux (Terminal)	`sha256sum file.pdf`	64-char hex + filename
macOS (shasum variant)	`shasum -a 256 file.pdf`	64-char hex + filename
PowerShell (export to CSV)	`Get-FileHash .\* -Algorithm SHA256 | Export-Csv hashes.csv`	CSV file with path, algorithm, and hash columns

3.2 Phase 2: Storage and Protection (CUI Considerations for Hash Logs)

Your hash manifest is itself a sensitive document. While the hash digests themselves aren't CUI, the manifest reveals the structure, naming conventions, and composition of your evidence repository. Treat it with appropriate care.

Key storage requirements for hash logs and manifests:

- **Access Control:** Store hash manifests in the same access-controlled repository as the artifacts themselves. Restrict write access to the compliance officer or the designated evidence custodian role.
- **Immutability:** Where possible, use write-once or append-only storage for hash manifests. Cloud storage services with object versioning and deletion locks (such as S3 Object Lock in COMPLIANCE mode or SharePoint with version history and checkout enforcement) provide good baseline controls.
- **Segregation of Duties:** The individual who collects an artifact shouldn't be the sole person responsible for generating and recording its hash. A two-person integrity check, where a second reviewer validates the hash at the time of collection, is a best practice for high-stakes evidence items.
- **CUI Boundary Awareness:** If your evidence artifacts contain CUI (for example, a system configuration file that includes network topology of a covered

system), the hash manifest indirectly references CUI and should be stored within your CUI enclave, not on general-purpose file shares.

- **Retention:** CMMC artifacts and their associated hash manifests must be retained for a minimum of six (6) years from the CMMC Status Date, as required by 32 CFR § 170.17(c)(4). This obligation applies to all certification levels and supersedes any shorter general federal records-retention guidance.

3.3 Phase 3: Presentation (Delivering a Hash Manifest to a Reviewer)

A C3PAO reviewer will expect your evidence to be organized, labeled, and verifiable. A hash manifest is the bridge between your internal evidence management practices and the reviewer's verification workflow.

A properly structured Hash Manifest includes the following fields for each artifact:

Manifest Field	Description and Requirements
Artifact ID	A unique identifier for the artifact. Use a consistent scheme: e.g., CMMC-SI-001, CMMC-AC-003.
Filename	Exact filename including extension, as it exists in the repository.
CMMC Control Reference	The CMMC 2.0 practice(s) this artifact supports. Example: SI.L2-3.14.2.
Collection Timestamp	Date and time the artifact was collected, in ISO 8601 format (UTC). Example: 2025-03-14T14:32:00Z.
Hash Algorithm	Always SHA-256. Document the algorithm explicitly so reviewers don't have to guess.

Manifest Field	Description and Requirements
SHA-256 Digest	The full 64-character hexadecimal digest string.
Collected By	Name and role of the individual who collected the artifact and generated the hash.
Verified By	Name and role of the second reviewer who independently verified the hash (if applicable).
Notes	Any collection context: tool version used, scope limitations, or known exceptions.

Deliver the manifest as a CSV or Excel file alongside the artifact package. Some organizations also include a signed PDF version of the manifest to provide an additional layer of authenticity. The Artifact Hash List file itself should be hashed to produce a Master Hash: a single SHA-256 digest of the entire manifest. This Master Hash is the value the C3PAO enters into the Hash Value field in the CMMC instantiation of eMASS under 32 CFR § 170.17(c)(4), creating a cryptographically verifiable chain from individual artifacts up to the final eMASS record.

Section 4: Tooling Analysis

The right tool for hashing depends on your environment, your team's technical depth, and how tightly you need the hash to integrate with your broader evidence management workflow.

4.1 Native OS Hashing: PowerShell and Terminal

Every Windows, macOS, and Linux system ships with a native hashing capability. These tools require no additional software, create no vendor dependencies, and produce universally verifiable output.

Windows PowerShell

PowerShell's Get-FileHash cmdlet is the recommended tool for Windows environments. It's scriptable, supports pipeline output, and produces structured objects that can be exported directly to CSV.

Basic usage: `Get-FileHash C:\Evidence\policy_v3.pdf -Algorithm SHA256`

Batch hashing to CSV: `Get-FileHash C:\Evidence\* -Algorithm SHA256 | Export-Csv C:\Manifests\manifest.csv -NoTypeInformation`

Pros of PowerShell hashing: no dependencies required, scriptable for automation, output integrates natively with other PowerShell workflows, and audit-friendly command syntax.

Limitations: requires configuring the PowerShell execution policy in some hardened environments; no built-in manifest template formatting.

macOS and Linux Terminal

SHA-256 on Linux: `sha256sum evidence_file.log`

SHA-256 on macOS: `shasum -a 256 evidence_file.log`

Batch hashing to a manifest file (Linux): `sha256sum /evidence/* > manifest.sha256`

The resulting .sha256 file can be verified by any recipient using the same tool: sha256sum --check manifest.sha256. This produces a pass/fail output for every file in the manifest.

4.2 GRC Tool Automated Hashing: Pros and Cons

Many GRC and compliance platforms purpose-built for CMMC and adjacent frameworks offer automated evidence collection with integrated hashing features. These tools reduce manual effort but introduce their own considerations.

Factor	Advantage	Limitation / Risk
Automation	Reduces manual effort; hashing is triggered automatically at collection.	Automation can mask the underlying algorithm. Verify that the platform uses SHA-256, not MD5 or SHA-1.
Documentation	Many platforms auto-generate structured evidence logs with timestamps and metadata.	The manifest format may not match C3PAO expectations. Export and reformat as needed.
Vendor Dependency	Consistent, repeatable process across team members.	If the platform changes its hashing implementation or you lose access, historical verification may be impossible.
Algorithm Transparency	Well-documented platforms publish their hashing specifications.	Some platforms don't clearly disclose the algorithm. Request documentation from the vendor before relying on it for CMMC evidence.
Independent Verification	Trusted platform hashes may be accepted by reviewers without re-verification.	A C3PAO may still require independently computed hashes for high-stakes artifacts. Don't rely solely on GRC-generated hashes without a manual spot-check process.

Recommendation: Use GRC automated hashing as your primary collection mechanism where available, but maintain the capability to independently hash and verify using native OS tools. Run a monthly spot-check comparing GRC-generated hashes against independently computed values for a sample of artifacts.

WARNING: CMMC-SPECIFIC GRC PLATFORMS AND THE LIMITS OF AUTOMATED HASHING

Several GRC platforms are purpose-built specifically for the CMMC community, designed around the 110 controls and 320 assessment objectives required for Level

2 certification. These platforms typically provide structured evidence storage, SSP automation, and secure export functions hosted on FedRAMP-compliant cloud infrastructure. When it comes to organizing and presenting compliance documentation, they generally do the job well.

What none of these platforms have publicly addressed is what security professionals call the collection-to-hash gap. This vulnerability class is formally known as a Time of Check to Time of Use (TOCTOU) condition: a well-documented category in computer security in which the state of a resource changes between the moment it is verified and the moment it is actually used. In this context, the gap exists between when a file leaves its source system and when the GRC platform computes its digest. No major CMMC-focused platform has published technical documentation describing exactly when in its ingestion pipeline a hash is computed, which algorithm is used, or whether hashing occurs at the moment of upload or in a subsequent background process. During that window, the file passes through a user's local machine, possibly via a downloads folder, desktop, or cloud sync directory, before being uploaded. Any modification that occurs in that window, intentional or accidental, will go undetected by the platform. TOCTOU conditions are documented in the MITRE Common Weakness Enumeration as CWE-367 and are recognized by NIST as a legitimate integrity risk in automated processing pipelines.

This is precisely why Organizations Seeking Compliance (OSCs) cannot treat GRC platform hashing as a substitute for understanding and performing manual hashing procedures. The CMMC standard requires demonstrable, verifiable evidence of integrity. A C3PAO is not obligated to accept a platform-generated hash as proof of integrity if the collection process that preceded it cannot be independently validated. If an OSC cannot explain what SHA-256 is, cannot demonstrate how to run Get-FileHash or sha256sum, and has no independently maintained hash manifest outside their GRC platform, they have a gap in their evidence-integrity posture, regardless of how polished their platform exports look.

The practical standard is this: your team should be able to hash a file, record the digest, and verify it again later using nothing more than the operating system already on their workstation. GRC platforms are an efficiency layer on top of that capability. They are not a replacement for it. An OSC that relies entirely on platform automation without understanding the underlying process is not in control of its evidence. They are trusting a vendor to manage it for them, and that trust is currently unverifiable on any platform in this market.

4.3 Third-Party Verification Procedures

When a C3PAO or your legal team needs to independently verify the integrity of artifacts you've provided, the verification workflow is straightforward but must be followed precisely.

Step 1: The reviewer receives the artifact file and the hash manifest.

Step 2: The reviewer independently runs the hash command against the received file using the algorithm documented in the manifest.

Step 3: The reviewer compares their computed digest against the digest in the manifest, character by character. A mismatch of even one character means the file has been altered.

Step 4: The reviewer documents the verification result, including the tool used, the computed digest, and the outcome, in their working papers.

For large evidence packages, reviewers may hash the entire directory archive (e.g., the ZIP file containing all artifacts) and compare that single digest against the cover memo. This is the hash-of-hashes approach and provides efficient verification without requiring per-file checks on every item in a large package.

Section 5: Process Workflow

A standardized Evidence Repository with mandatory hashing transforms ad hoc file collection into a defensible, audit-ready practice.

The following workflow establishes a repeatable process for building and maintaining a CMMC-aligned evidence repository. Each step is designed to be executed by an IT administrator or compliance officer without specialized forensic tools.

Step	Phase	Procedure
1	**Repository Setup**	Create the Evidence Repository directory structure: /CMMC-Evidence/ /Active/ (current review cycle artifacts) /Manifests/ (hash manifest files)

Step	Phase	Procedure
		/Archive/ (prior review cycles) /Templates/ (blank manifest CSV template) Apply access controls: read/write for Compliance Officer role; read-only for other authorized personnel.
2	**Pre-Collection Preparation**	Before collecting any artifact: a. Assign an Artifact ID using your naming convention (e.g., CMMC-AC-001). b. Open the Hash Manifest CSV template for the current review cycle. c. Document the control reference, expected artifact type, and collector name.
3	**Artifact Collection**	Collect the artifact using the appropriate method (export, screenshot, report download). Save to /CMMC-Evidence/Active/ using the naming convention: [ArtifactID]_[YYYY-MM-DD]_[description].[ext] Example: CMMC-AC-001_2025-03-14_AD-MFA-Policy.pdf
4	**Immediate Hashing**	Run the hash command immediately after saving the file. Do not open, move, or rename the file first. PowerShell: Get-FileHash .\CMMC-AC-001_2025-03-14_AD-MFA-Policy.pdf -Algorithm SHA256 Record the full 64-character digest in the manifest. Record the timestamp.
5	**Manifest Entry**	Complete all required fields in the Hash Manifest CSV: Artifact ID, Filename, CMMC Control Reference, Collection Timestamp (UTC), Algorithm (SHA-256), Digest, Collected By, Notes. Save and close the manifest file. Do not leave the manifest open in an editable state when not actively in use.
6	**Second-Party Verification**	For critical artifacts (policies, SSP, plans of action), a second reviewer should: a. Independently compute the hash of the artifact file. b. Compare it against the recorded digest in the manifest.

Step	Phase	Procedure
		c. Record their name and verification timestamp in the Verified By field.
7	**Repository Integrity Check**	Conduct weekly integrity checks during active collection periods: PowerShell: Get-FileHash /CMMC-Evidence/Active/* -Algorithm SHA256 | Compare-Object (Import-Csv /Manifests/manifest.csv) Investigate and document any discrepancies immediately. A mismatch indicates either an unauthorized modification or a collection process failure.
8	**Pre-Delivery Package Preparation**	Before delivering artifacts to a C3PAO: a. Run a final full-repository integrity check. b. Export the final manifest log as a plain-text file listing each artifact name, its SHA-256 digest, and the algorithm used. This is your Artifact Hash List. c. Generate a Master Hash: run SHA-256 against the Artifact Hash List file itself. This single digest is the hash-of-hashes and is the value your C3PAO will enter into the Hash Value field in the CMMC instantiation of eMASS per 32 CFR § 170.17(c)(4). d. Provide the C3PAO with three items: the artifact files, the Artifact Hash List file (filename entered into the Hashed Data List field in eMASS), and the Master Hash value (entered into the Hash Value field in eMASS). e. Note: the C3PAO's Quality Assurance Professional performs the eMASS upload. The OSC does not upload directly to eMASS for Level 2 certification assessments.
9	**Post-Review Archival**	After the Readiness Engagement is complete and the C3PAO certification assessment has concluded: Move the entire Active folder, its Artifact Hash List, and the Master Hash record to /Archive/[ReviewCycle-YYYY]/ Apply write protection or legal hold designation. Retain for a minimum of six (6) years from the CMMC Status Date, per 32 CFR § 170.17(c)(4).

Section 6: Compliance Check

File hashing provides direct, documentable support for specific NIST SP 800-171 controls. The mapping below is what a C3PAO will test, and your hash manifest is the evidence that closes the loop.

The table below maps the hashing workflow described in this document to the specific NIST SP 800-171 Rev. 2 controls it supports. Use this mapping to annotate your System Security Plan (SSP) and evidence packages.

CMMC Practice	NIST 800-171 Control	How Hashing Supports This Control
SI.L1-3.14.1	3.14.1	**Identify, report, and correct information and information system flaws in a timely manner.** Hashing verifies that system-generated flaw reports and scan outputs haven't been modified between collection and review. A hash mismatch on a vulnerability report is itself a reportable integrity event.
SI.L2-3.14.2	3.14.2	**Provide protection from malicious code at appropriate locations.** Hash manifests for antimalware configuration exports and scan logs demonstrate that protection records are authentic and unmodified, supporting verification of the control's continuous operation.
CM.L2-3.4.1	3.4.1	**Establish and maintain baseline configurations and inventories of organizational systems.** Hashing baseline configuration files (GPO exports, firewall rulesets, hardening checklists) creates a cryptographic record of the baseline at a specific point in time. Drift detection is enabled by comparing current hashes against the baseline manifest.
CM.L2-3.4.3	3.4.3	**Track, review, approve, and log changes to organizational systems.**

CMMC Practice	NIST 800-171 Control	How Hashing Supports This Control
		Change management artifacts (change request forms, approval records, pre-change and post-change configuration exports) should each be hashed to prove the change record is complete and unmodified.
AU.L2-3.3.1	3.3.1	**Create and retain system audit logs and records to enable monitoring, analysis, investigation, and reporting.** Exported audit log files should be hashed at the time of export. This proves to a reviewer that the log contents haven't been selectively edited. This is particularly important for access control and authentication event logs.
AU.L2-3.3.2	3.3.2	**Ensure that the actions of individual users can be uniquely traced to those users.** Hash verification of audit logs supports the non-repudiation of user actions. If a log file's hash doesn't match its manifest entry, the chain of accountability for that log is broken and must be investigated.
MP.L2-3.8.6	3.8.6	**Implement cryptographic mechanisms to protect the confidentiality of CUI during transmission.** When CUI-containing artifacts must be transmitted to a reviewer, hashing provides end-to-end integrity verification: the sender documents the hash, and the receiver verifies it upon receipt, confirming that transmission didn't alter the file.
SC.L2-3.13.10	3.13.10	**Establish and manage cryptographic keys for cryptography employed in organizational systems.** Documentation of your cryptographic key management procedures, including approved algorithm lists that specify SHA-256, supports compliance with this control. Your hash manifest itself demonstrates operational use of approved cryptographic mechanisms.

6.1 Documenting Hash Implementation in Your SSP

Your System Security Plan must describe how hashing is implemented as an integrity control. The following language can be adapted for your SSP narrative:

[Organization] implements cryptographic file hashing using the SHA-256 algorithm (FIPS 180-4) for all evidence artifacts collected in support of CMMC readiness activities. Hash digests are recorded in a structured Hash Manifest at the time of artifact collection. The manifest is maintained in the access-controlled Evidence Repository alongside the artifacts it references. Integrity checks are performed weekly during active collection periods and prior to all external deliveries. Hash verification procedures are documented in [Policy Name], Section [X.X].

6.2 Quick Reference: Compliance Readiness Checklist

- SHA-256 is the only hashing algorithm in use for CMMC evidence artifacts.
- Hash is generated immediately at artifact collection, before any file operations.
- Hash Manifest includes all required fields: Artifact ID, filename, control reference, timestamp, algorithm, digest, collector, and verifier.
- Hash Manifest is stored in an access-controlled repository with write restrictions.
- Weekly integrity checks are conducted and documented during active collection periods.
- Pre-delivery package integrity check is performed before any external delivery.
- SSP narrative documents the hashing process, tool, and algorithm explicitly.
- Evidence archives are retained for a minimum of six (6) years from the CMMC Status Date per 32 CFR § 170.17(c)(4), with write protection or legal hold designation applied.
- Team members responsible for evidence collection have received training on hash procedures.
- GRC platform hashing algorithm has been verified as SHA-256 with vendor documentation on file.

About the Author

David W. Koran is the founder and principal consultant at David Koran & Associates, a CMMC compliance consulting firm serving Defense Industrial Base contractors and the legal counsel who represent them. He is a CMMC Registered Practitioner with over 30 years of experience in IT and cybersecurity, and an Associate Member of the ABA Section of Public Contract Law.

This document is prepared for educational and client readiness purposes only. It does not constitute legal advice. Consult qualified legal counsel for matters involving False Claims Act liability, contractual obligations, or regulatory enforcement.