PORTABLE FILE INTEGRITY FOR AIR-GAPPED ENVIRONMENTS

How trust can survive the crossing into disconnected, tactical, and zero-trust domains

Abstract v1.3

For decades, a file's name has had no cryptographic connection to its content. Hashes protect the bytes and signatures prove the signer, but neither binds the name users and systems actually rely on. VeraFile closes that gap by deriving a compact, human-readable Validation Integrity Key from a file's contents, name, and metadata and embedding it in the file name, so the file proves its own integrity locally - even inside an air gap, with no network, reach-back, or enterprise infrastructure.

VeraSec Technologies

About VeraSec Technologies LLC

VeraSec Technologies is a pioneer in Artifact Identity Security *(AIS)*, focused on moving the security context from the infrastructure directly to the file itself. While traditional security models rely on centralized systems to establish and maintain trust, VeraSec's patent-pending VeraFile technology empowers files to carry their own verifiable identity - preparing them for an actual Zero Trust world.

Operating at the forefront of Portable File Identity *(PFI)* technology, VeraSec invented the Validation Integrity Key *(VIK)* process. This innovation patches a fundamental vulnerability in traditional PKI that has persisted for decades. By cryptographically binding a file's content and metadata into a compact, human-readable identifier embedded directly within the artifact, VeraFile helps organizations set up durable, portable trust in every digital asset, signed or unsigned - wherever it travels.

About The Publisher

Brian Hajost (bhajost@verasec.net) is a cybersecurity entrepreneur and technologist with more than 30 years of experience in information security, compliance automation, and trusted computing systems.

As the founder of SteelCloud, he pioneered the automation of DISA STIG and CIS benchmark compliance for the DoD, federal agencies, and enterprise customers. Brian holds 15 patents in information technology and mobile security. In 2022 he was named one of the *"Top 10 Most Influential People in Cyber Security"* by CIO Views.

His current work focuses on solving what he describes as the Artifact Identity Problem - the absence of durable identity for files as they move across distributed infrastructures and untrusted networks.

Disclosure

Portable File Integrity for Air-Gapped Environments

How trust can survive the crossing into disconnected, tactical, and zero-trust domains

Executive Summary

For decades, computing has carried a quiet flaw: a file's name has no cryptographic connection to its content. A file can be named anything. A content hash protects the bytes but says nothing about the name. A digital signature signs the content or a byte range - not the file's external name. As a result, a file can be renamed, relabeled, or moved out of context, and the hash still matches and the signature still verifies. Nothing flags it.

This gap is not academic. In real workflows, the file name carries the operational meaning. It tells a user or an automated process what something is: its version, its classification, its mission, its approval state, its destination. Renaming a file, accidentally or deliberately, can mislead a person, an import process, an automation pipeline, or a loader, even when the content is untouched and a valid signature is present. A stale build relabeled as the current patch, a test file relabeled as final, or a validly signed but dangerous utility relabeled as an approved tool all pass today's conventional integrity checks.

VeraFile closes this gap by making the file name part of the trust object. It derives a compact, human-readable Validation Integrity Key (VIK) from the file's content, name, and metadata, and embeds it directly into the file name. The file carries its own validation reference wherever it goes. A VeraFile validator recomputes the VIK locally and determines whether the file's content-and-name identity is still the one that was sealed - with no enterprise connection, no cloud service, no reach-back, and no PKI chain validation required merely to answer that question.

This is especially valuable in air-gapped and tactical environments, where the systems that normally provide assurance - repositories, endpoint agents, SIEMs, identity providers, certificate revocation services - are absent or unreachable, and where scarce, contested communications should not be spent on routine file checks. VeraFile lets the file prove itself locally.

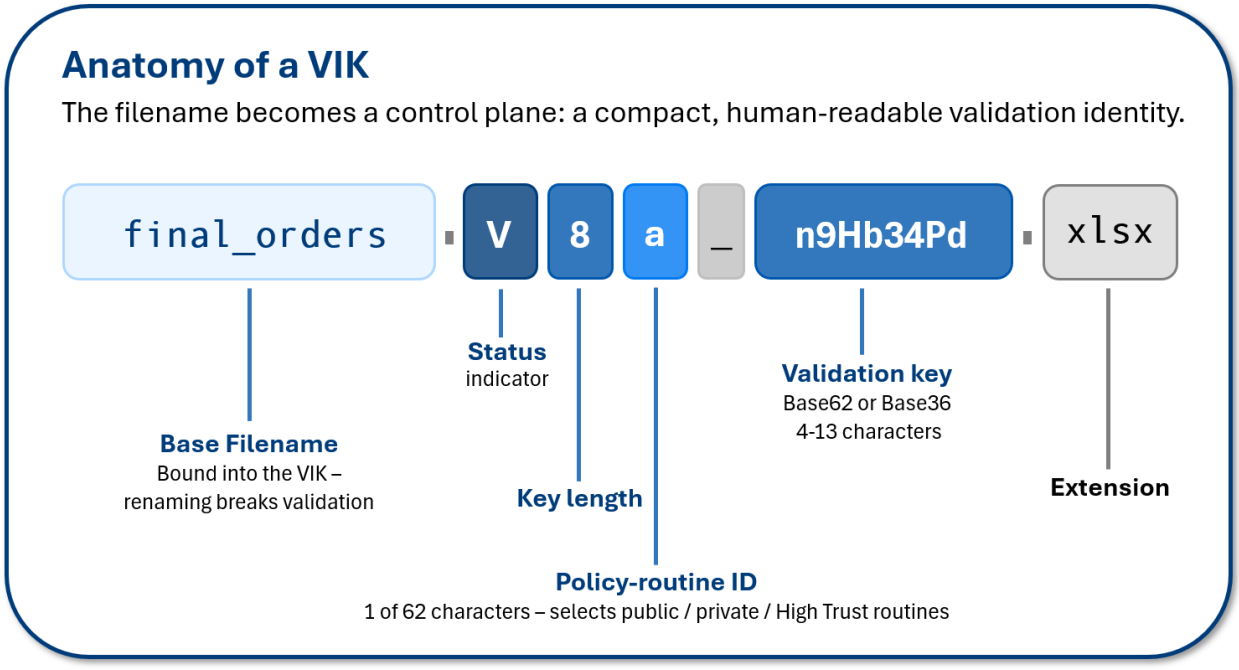
VeraFile operates in three trust modes, and the paper is explicit about what each guarantees:

- **Public mode** - the derivation routine is open, and anyone can recompute the VIK. This delivers portable, infrastructure-free detection of any change or rename that is not re-sealed: corruption, truncation, tool-induced changes, stale or mislabeled files, and naive substitution.

- **Private mode** - the routine incorporates organization-held secret material, so outsiders cannot reproduce a valid VIK even with the file and its visible name.
- **High Trust mode** - VIK calculation is split between local and central protected components, adding governance, secret rotation, and brute-force monitoring for the highest-assurance use.

Because sealing and validation are fast and infrastructure-free - approximately 500,000 small files per hour on a single thread - VeraFile is built to protect files in bulk, not only the small fraction an organization decides in advance to treat as sensitive. That makes it practical to seal everything and validate everything, rather than pre-triaging which files deserve integrity.

The result is a file-centric trust model that complements, rather than replaces, hashing, signing, and encryption, and that survives the crossing into environments where the network cannot be trusted, cannot be reached, or does not exist.

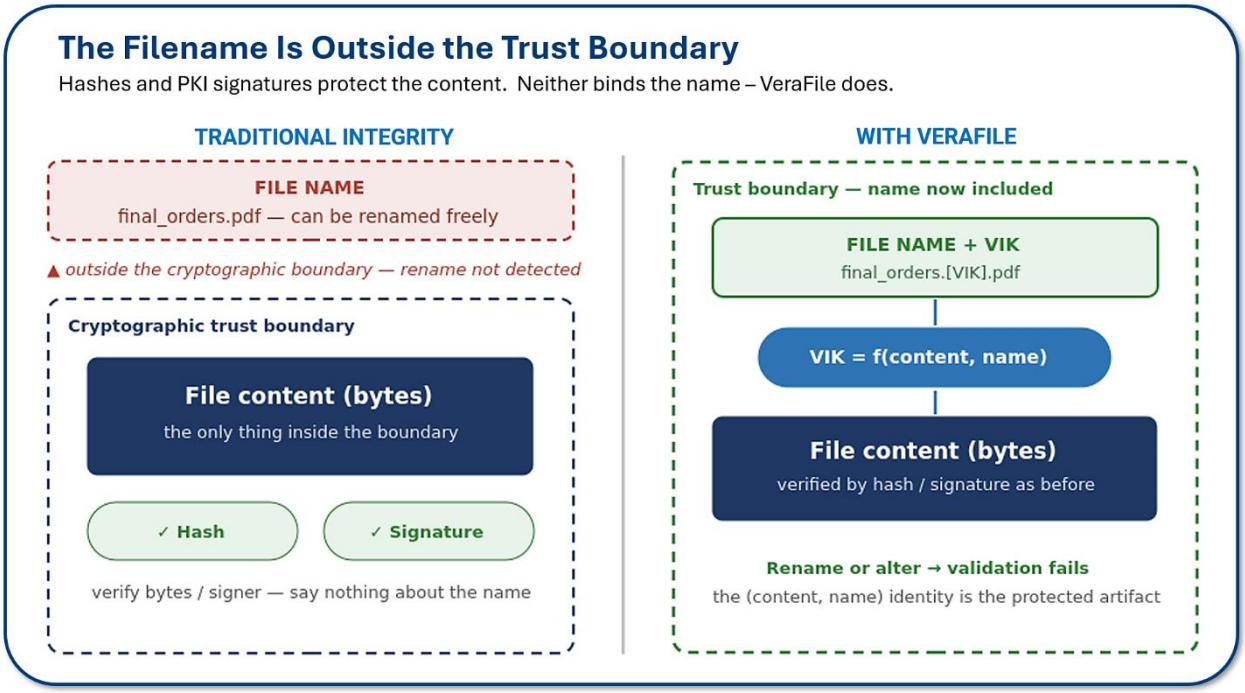


The Gap VeraFile Closes: *The File Name Is Outside the Trust Boundary*

Every mainstream integrity technology stops at the file's content.

A cryptographic hash answers one question: *did the bytes change?* It does not bind the bytes to a filename. The same content hashes to the same value whether the file is called final_orders.pdf or old_draft.pdf.

A digital signature answers a related question: *was this content signed under a particular key?* In Authenticode, PDF, CMS, and GPG signing, the signature covers the document or a byte range - not the operating-system file name. A validly signed file can be renamed to anything, and the signature still verifies.



So, the file name, the thing humans and name-dependent systems actually rely on, sits *outside* the cryptographic boundary. This produces two persistent classes of failure:

Unintentional Havoc: Files are renamed constantly and casually: final, final_v2, FINAL_USE_THIS, copy (3). Tools rename. Pipelines ingest the wrong-named file. None of this is malicious, and none of it is caught by a hash or a signature, because neither references the name.

Intentional Havoc: An adversary often does not need to alter content. Relabeling a benign but wrong file to look authoritative, current, or approved can be enough to mislead a user, a workflow, an evidence system, or an automated import. This is acutely dangerous with program files: a binary may carry a perfectly valid signature yet be the wrong artifact for the

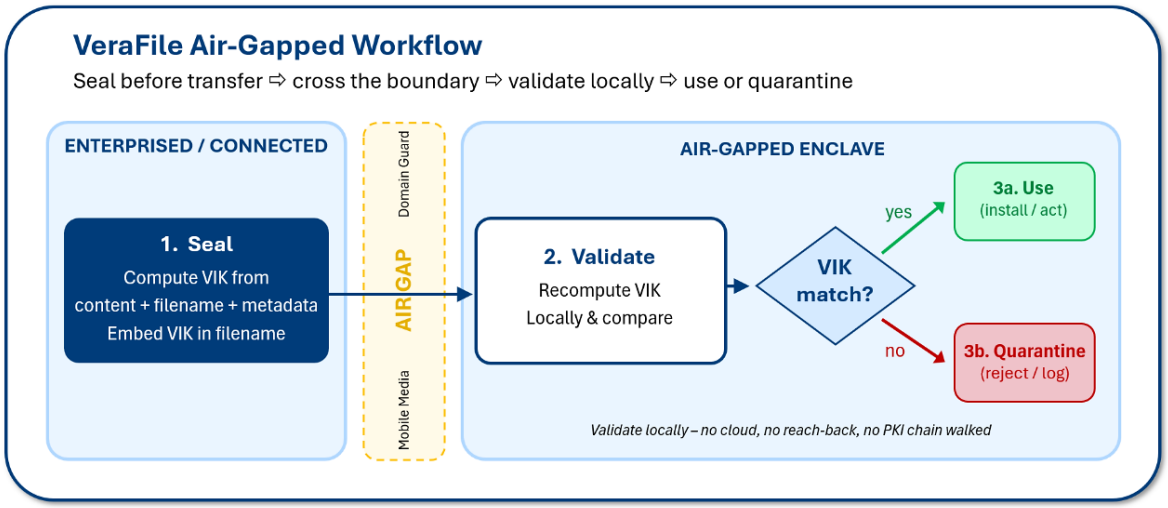
context - an old version with a known vulnerability, a powerful system utility relabeled as an approved tool, or a signed file whose revocation status cannot be checked offline. Allow-listing by signature says, *"trusted publisher, allow,"* while the name quietly lies about *which* artifact this is.

VeraFile binds the name into the integrity calculation. The approved artifact is the *(content, name, metadata)* triple, not the content alone. Change any of them without re-sealing (which renames the file), and validation fails. This is the capability that hashes and signatures do not provide.

Why This Bites Hardest in Air-Gapped Environments

Air gaps reduce risk by separating sensitive systems from external networks. They do not eliminate the need to move files - updates, configuration, mission orders, intelligence products, drawings, logs, patches, scripts, images, and operational data all cross the boundary routinely.

The moment a file crosses, much of the enterprise's assurance apparatus disappears. Context is lost; the file may no longer be associated with the system, user, repository, or workflow that produced it. Metadata is stripped by transfer tools, cross-domain guards, and manual handling. Validation services - hash databases, scanners, signing authorities, revocation services, identity providers, are unreachable. Automation gives way to manual procedures, and manual procedures do not scale to thousands or millions of files.



In that setting, the file name often becomes the *primary* surviving signal of what a file is - which is exactly the signal that conventional integrity leaves unprotected. VeraFile makes that surviving signal trustworthy, and **it does so without requiring the air-gapped side to recreate the enterprise security stack.**

The VeraFile Model: *Validation that Travels with the File*

At the center of VeraFile is the Validation Integrity Key (VIK). The VIK is a compact, file-system-compatible, human-readable identifier derived from the file's content, name, and metadata, plus policy-selected inputs (extension, metadata, signer identity, container membership, and so on). It is embedded in the file name and travels with the file across systems, repositories, devices, and security boundaries.

To validate, a VeraFile validator strips the existing VIK, normalizes the base name, recomputes the VIK from the received file under the applicable policy, and compares it to the embedded value. If they match, the file's content-and-name identity is intact. If they do not, the file is flagged as changed, renamed, corrupted, substituted, or otherwise inconsistent with its sealed identity.

This turns the file name into a **control plane** rather than a mere label. It can carry not only an integrity value but also encoded policy fields - validation status, key length, policy-routine identifier, organizational identifier - so that a short token in the name can reference an entire customer-defined protection routine. A single Base62 character can select from as many as 62 distinct customer-centric routines.

The decisive property for disconnected operations is that all of this is verifiable **locally**. The validator does not query a server, retrieve a manifest, validate a PKI chain, or reach the cloud merely to determine whether the file has changed. The validation capability is built into the file name - lightweight, portable, and infrastructure-free.

Three Trust Modes - *Exactly What Each Guarantees*

VeraFile supports three architectural modes, which can coexist in a single deployment. The distinction matters, because each mode defends a different threat and was developed for a different implementation.

Public Mode: ***Portable Integrity and Anti-mislabeling***

The derivation routine is open and standardized. Anyone with the file and its name can recompute the VIK; the file proves itself by deterministic recomputation, with no proprietary server, key store, or registry.

What this guarantees: detection of any change or rename that is not re-sealed - accidental corruption, truncation, encoding damage, tool-induced changes, stale and mislabeled files, and substitution by anyone who does not re-seal. Because a tampered file must carry a new VIK, it presents as a different filename and is therefore caught by any recipient who

knows the name they were supposed to receive. The human-readable VIK makes that expected name something you can put on a transmittal sheet, send in an email, or hold in an operator's memory - no infrastructure required.

The honest boundary: because the routine is public, a knowledgeable party can edit a file and compute a fresh, internally consistent VIK. A recipient doing pure self-consistency checking, with no expectation of what the name should be, cannot distinguish that re-sealed file. Public mode is therefore an integrity and anti-mislabeling layer, not, on its own, a defense against a determined adversary. For that, use private or High Trust mode.

Private Mode: *Substitution Resistance*

The derivation routine incorporates organization, tenant, domain, or policy-specific secret material (a private salt or pepper, a protected reduction routine, rotating secrets with legacy validation). An outsider holding the file and its visible name cannot reproduce a valid VIK.

This converts the VIK from a public integrity marker into an organization-controlled trust artifact, and it is where adversarial claims belong: a tampered or substituted file cannot be made to validate without the secret, and the check still runs locally and offline on systems that hold the routine. This is the mode for classified, controlled, partner, supply-chain, and program-specific use.

High Trust Mode: *Governed, Monitored, Highest Assurance*

VIK creation and validation are split between a local component and a central protected service. The local side hashes the file and collects inputs; it can send only derived or anonymized values (content hash, name hash, size hash, policy identifier) so that filenames and content need not leave the enclave. The central service applies a protected routine, returns the VIK or result, logs and throttles requests, detects brute-force patterns, enforces routine authorization, and manages secret rotation.

This is the right mode wherever a short, human-readable tag could otherwise invite a guessing attack, because the central service can monitor and block exactly that behavior.

A Note on VIK length

A VIK is 4–13 Base62 or Base36 characters. Length is chosen against the base name population, not against the world: collisions among same-named files are vanishingly unlikely even at modest lengths. But length interacts with the threat model. Short keys should be reserved for low-stakes or collision-only purposes. Longer VIKs should be employed for sensitive data or data that is subject to adversarial attack.

A single validator handles multiple routines concurrently. Because each VIK encodes its own length and policy-routine identifier, one validator can process a mixed population of files - varying VIK lengths, and multiple public, private, and High Trust routines - choosing the correct routine for each file.

Base62 Character Set *(A-Z, a-z, 0-9)*			
VIK Length	**Entropy Bits**	**Collision Risk**	
		1,000 Files	**10,000 Files**
4	23.8	3.32%	96.6%
6	35.7	1 in 113,700	1 in 1,100
8	47.6	1 in 437M	1 in 4.4M
10	59.5	1 in 1.7T	1 in 16.8B
12	71.5	1 in 9,000T	1 in 64.3T

Sealing Signed and Encrypted Files Without Opening Them

Because the VIK is computed over the file's bytes, name, and metadata, VeraFile can seal a file **without opening it** - including PKI-signed files, encrypted files, and classified files. No decryption, no signature validation, and no file parsing is required to seal a file or to check the validity of a VIK.

This integrates cleanly with existing protections:

- For **signed files**, VeraFile adds the name-binding the signature itself lacks. The signature continues to prove *what was signed*; the VIK adds *that this named artifact is the one that was sealed*. This directly addresses the validly-signed-but-wrong-artifact problem: a signed binary relabeled to impersonate an approved tool, or an outdated signed installer relabeled as the current patch, fails VeraFile validation even though its signature is genuine.
- For **encrypted files**, VeraFile lets a recipient confirm “this is the same encrypted artifact under the same name” without holding any key - useful for custody and transfer assurance of opaque payloads.

VeraFile does not replace signing or encryption. It supplies the portable, name-bound integrity layer those technologies were never designed to provide.

Protecting Containers: *ZIP Archives and PDF Envelopes*

Air-gapped environments often receive packages, not single files, and containers create their own integrity problems. Two ZIP files can hold the same visible files yet differ in compression, ordering, headers, timestamps, comments, and directory structure, so a binary hash of the container cannot easily separate meaningful tampering from benign repackaging. Conversely, a container can look intact while its contents, names, or structure have changed.

VeraFile applies integrity at multiple layers:

- **Child-file validation** - each file inside the container is individually validated against its content, name, metadata, and policy.
- **Manifest validation** - a machine and human-readable manifest lists expected internal names, VIKs, sizes, hashes, and policy identifiers.
- **Container validation** - the ZIP or PDF envelope itself receives a VIK (in the name and/or container metadata), validating the whole archive as a single portable artifact.

Together these let VeraFile distinguish at least four conditions that ordinary container hashing blurs: the container changed but its children did not; one or more children changed; internal names or paths changed; or membership and structure changed. PDF secure envelopes, portfolios, and packages are handled with the same layered approach, binding embedded-file name, content, and envelope context together and distinguishing wrapper changes from payload changes.

The container is only one layer, not the foundation. Because each file inside was sealed with its own VIK, its integrity is intrinsic to the file rather than to the package. When the container is used purely as a transport wrapper, the extracted files continue to validate individually in later workflows - long after the container itself is gone.

What VeraFile Is, and Is Not

A credible integrity claim states its threat model. VeraFile's is straightforward. VeraFile is not a substitute for signing, encryption, scanning, or access control. It is the portable, name-bound integrity layer that those controls do not provide, designed to keep working when infrastructure is absent.

Architecture and Workflow

A typical air-gap deployment has four stages:

Seal before transfer. Files (and containers and signed or encrypted artifacts) are sealed at the point of release or export. For high-assurance flows, sealing uses a private or High Trust routine.

Cross the boundary. The file moves through the organization's approved process - removable media, cross-domain guard, scanning station, or physical delivery. Even if metadata is stripped and context is lost, the VIK travels with the file.

Validate locally. Inside the enclave, a validator recomputes the VIK and compares. No reach-back, cloud, or PKI service is consulted to determine whether the file changed.

Use or handle the exception. Validated files proceed; failures are quarantined, rejected, logged, or escalated by local policy. For high-risk flows, validate at ingress *and* again immediately before execution, installation, or mission use.

Use Cases

Tactical mission packages. A planning environment seals a package - maps, imagery, orders, target data, configuration, as a VeraFile container before transfer. The deployed unit validates locally whether the package is intact, whether the archive was reconstructed, and whether each file remains valid. No satellite reach-back is consumed for routine validation.

Software updates for disconnected systems. Patches and security content are sealed before transfer and validated inside the enclave, giving administrators a local way to confirm the package is the same artifact approved for installation - and, for ZIPs, that it was not reconstructed.

Industrial control and critical infrastructure. Firmware, PLC logic, configuration, and vendor files arrive through manual transfer into network-separated environments. A file sealed before transfer can be validated before use, catching renamed, altered, or substituted files where safety and uptime are at stake.

Forensics and chain of custody. Files and evidence containers carry portable, name-bound validation identities, reducing the administrative burden of detached hash lists while complementing formal custody documentation.

Intra-enclave data. VeraFile is not only for files crossing the boundary. An air-gapped environment can seal and validate the data it produces and uses internally - logs, analyst work products, generated reports, configuration, and collected data, giving it a lightweight integrity layer where there is often no SIEM, repository, or endpoint tooling of its own. Both sealing and validation are local, so periodic sweeps can surface silent corruption, accidental overwrites, or version confusion long after the data was created; where deliberate internal modification is a concern, a locally held private routine extends this beyond accidental change.

Classified and compartmented enclaves. Files validate inside the enclave without reach-back to the originating system - preserving the separation principles that make the enclave secure while letting users confirm that what they received is what was approved.

Built for Volume: *Seal Everything, Not the 2–5%*

Selective integrity is a compromise forced by cost. When sealing and validating are slow, expensive, or infrastructure-dependent, organizations protect only the files they judge to matter, and only the final versions of those files - often a few percent of the total. They leave the rest unverified. Both attackers and accidents live in that unverified majority: work-in-process files and only the finalized files no one thought were important enough to justify protection.

VeraFile is built to remove that compromise. A single thread seals approximately 500,000 small files per hour, operating at the speed of the surrounding storage and pipeline rather than gated by network calls, and throughput scales with available cores and I/O. Because validation is local and infrastructure-free, it runs at comparable speed wherever the files land.

That changes the security posture. Instead of pre-deciding which files deserve integrity, an organization can seal everything - every artifact in a repository, every file in a transfer, every object written to removable media, and validate everything at ingress and before use. There is no “unimportant file” blind spot, no triage step to get wrong, and no class of files silently exempt from the trust model. Bulk sealing of repositories, staged-media validation, full-package verification, and periodic integrity sweeps across millions of files become friction-free routine operations rather than aspirations.

Why This Matters for Zero Trust

Zero Trust is usually framed around users, devices, networks, and applications, but files are the most persistent trust gap. They move across domains, outlive their origin systems, and are copied, renamed, archived, compressed, and reintroduced into sensitive environments. In air-gapped settings, most Zero Trust enforcement points are simply absent.

VeraFile extends Zero Trust to the file itself: a file is not trusted because it arrived from a trusted place, but because it validates against its own portable, name-bound identity. In private and High Trust mode, that validation holds even against an adversary who controls the transfer path.

Comparison With Traditional Approaches

Approach	Strength	Limitation in air-gapped use
Enterprise repository controls	Strong before transfer	Context lost after the crossing
Endpoint security agents	Useful on managed systems	Often absent inside the enclave
Detached manifests	Validate file sets	Separate from files; manual; fragile at scale
Full cryptographic hashes	Strong content identifiers	Not name-bound; not human-friendly
PKI signatures	Strong content/key proof	Sign content, not name; chain/revocation often unavailable offline
Cloud validation services	Scalable when connected	Unusable without connectivity
VeraFile	Portable, local, name-bound validation across modes	Protects the integrity of all file types - signed and/or encrypted with portable VIK that survives file movement

VeraFile's advantage is not that it replaces every control. It is that it binds the name to the content - the gap the others leave open and preserves a practical integrity control when the others are unavailable.

Conclusion

The oldest unsolved problem here is not proving a file was once trusted inside the enterprise. It is that a file's name has never been connected to its content - so a renamed or relabeled file passes every conventional check. Air-gapped environments expose the consequences sharply because the name is often the only signal that survives the crossing.

VeraFile binds the name to the content and lets that binding travel with the file. In public mode it provides portable, infrastructure-free integrity and anti-mislabeling. In private and High Trust mode it provides tamper and substitution resistance that holds even against an adversary, still verifiable locally and offline. It seals signed and encrypted files without opening them, protects containers at multiple layers, and turns the file name from an unprotected label into a portable control plane. And because it operates at infrastructure speed, it is built to do this for every file - not only the small fraction an organization can afford to single out today.

The future of secure file movement is not only encryption, signing, scanning, and access control. It is portable, name-bound integrity. That is what VeraFile delivers.