SECURITY & AUDIT WHITEPAPER · V1.1

# A working framework for auditing AI tools that touch confidential documents.

The four security claims SealedBrief makes about its own design, with step-by-step instructions for verifying each one on the live binary — and ten questions to put to any AI vendor before you trust them with your client files.


AUTHOR
SealedBrief engineering team


AUDIENCE
Lawyers · Therapists · Accountants


DATE
June 2026


PAGES
5

§1 — ABSTRACT

# Why this exists.

Every privacy-positioned AI product on the market makes claims about where your data goes and what happens to it. Most of those claims cannot be verified by the buyer. This document is a working framework for changing that.

The framework has two halves. The first is four security claims SealedBrief makes about its own design, with step-by-step instructions for verifying each one on the live binary using tools available on any modern Linux or macOS workstation. The second is a ten-item checklist a buyer can put to any AI vendor — cloud or local — to distinguish honest claims from marketing assertions. SealedBrief is built to satisfy every item; competitors are scored against the same list so the comparison is structural, not promotional.

## The regulatory reality

Several professional regimes bind the typical SealedBrief buyer. The three U.S. regimes below are illustrative, not exhaustive; the same third-party-transmission exposure shows up under Brazil's LGPD and professional-secrecy duties (*sigilo profissional*) and under the EU/UK GDPR for health data and legal-privilege material. None of them are abstractions:

**Lawyers.** Client confidentiality under *ABA Model Rule 1.6* obligates a lawyer to make reasonable efforts to prevent unauthorized disclosure of privileged information (Rule 1.6 Cmt. 18), and the duty of competence extends to the technology a lawyer uses (Rule 1.1 Cmt. 8). *ABA Formal Opinion 512* (2024) is the controlling guidance on generative AI: it requires lawyers to understand where client data goes and to obtain informed consent before exposing it to a third party. Cloud LLM providers are third parties. Pasting a deposition transcript into a third-party API routes that material through the vendor's servers, contractors, and logging pipeline — a chain the lawyer cannot audit and rarely fully understands.

**Mental health practitioners.** PHI shared with a cloud AI provider triggers the Business Associate Agreement (BAA) regime under *HIPAA*'s Privacy and Security Rules. A BAA must be in place; the BAA is enforceable; and most consumer-facing AI tools simply do not offer one. The clinician who pastes a treatment plan into a chatbot to "just rephrase a paragraph" crosses the regime without realising it.

**Tax and accounting practitioners.** *IRS Pub 4557* and the *Gramm-Leach-Bliley Safeguards Rule* require a written information security program covering client PII (SSNs, financial records, correspondence). Third-party AI tooling triggers vendor-due-diligence obligations the small practice rarely has staff to discharge.

**SealedBrief is a tool, not a compliance program.** It removes the third-party-transmission exposure that the regimes above turn on — but you remain responsible for confidentiality, competence, and supervision; for your written information-security program; and for your own vendor and risk analysis. Nothing in this document is legal advice.

## The local-first thesis

Instead of accepting any of the above, run the AI on the practitioner's own machine. The architectural guarantee is "no data leaves the device"; the verification framework in §2 below lets you confirm that guarantee yourself, without trusting the vendor's word for it. The product is local-first by construction, not by promise.

§2 — FOUR CLAIMS, FOUR AUDITS

# Verify the design without trusting the vendor's word.

Each claim below is checkable on the live binary in under fifteen minutes. Tools required: `tcpdump` or `nethogs` (network), `ent` (entropy), and `gcore` (memory) — plus the SealedBrief audit scripts in `scripts/security/` of the published source repository, which wrap each step so the result is the same regardless of who runs it.

### 01 Zero network traffic from the document-handling process.

NETWORK AUDIT

**Contract.** Two operating-system processes run when SealedBrief is open: the Presentation Plane talks to our licence server (only) for activation and update checks; the Compute Plane handles your documents and never opens a network connection.

**How to audit.** Start the app. Identify the Compute Plane PID with `pgrep -af compute_service.py` (the process is named `Compute`). Capture its egress with the shipped script — positional arguments, `<pid> <output.pcap>`: `sudo ./scripts/security/capture_compute_egress.sh $(pgrep -f compute_service.py) compute_egress.pcap`. Ask any question, then analyse: `python scripts/security/analyze_egress_pcap.py compute_egress.pcap` — it asserts zero outbound packets from the Compute Plane. A lightweight live alternative is `sudo nethogs -p $(pgrep -f compute_service.py)`; expect zero non-loopback connections. Anything else is a finding.

### 02 Encryption at rest is real, not just claimed.

ENTROPY AUDIT

**Contract.** SealedBrief writes its encrypted index and metadata to a SQLCipher database at `data/core.db` (relative to the app's data directory), with vector embeddings stored separately under `data/lancedb/`. The `core.db` file must look like random ciphertext to a hex dump. We pin the entropy floor at 7.5 bits per byte across the encrypted file.

**How to audit.** Run `ent data/core.db`. Expect entropy ≥ 7.5 bits/byte. The shipped, authoritative proof is `pytest tests/security/test_core_db_entropy.py -v`, which asserts Shannon entropy > 7.5 bits/byte on `core.db` plus header and sentinel-substring counter-controls. Optional spot-check: open the file in a hex editor and confirm no ASCII sentences, no `%PDF` headers, no recognisable structure. Below the floor → finding.

## 03 After you lock the vault, no copy of the key bytes we control survives in memory.

KEY HANDLING

**Contract.** Your 256-bit master key lives in the OS keychain (libsecret on Linux, Keychain on macOS) and is read into RAM only while your vault is unlocked. For the raw 32-byte form of the key SealedBrief handles directly, we ask the OS to pin its memory page out of swap (`mlock`, best-effort — on a machine that refuses we log a warning rather than fail silently) and we overwrite those raw bytes with zeros (`explicit_bzero`) the moment we finish deriving your per-record keys via HKDF-SHA256. We will *not* pretend a dump of the running process finds nothing: while the vault is unlocked the key is genuinely in use, so SQLCipher's open database engine and the immutable hex-text form of the key can each appear in a live dump. Our guarantee is narrower and testable — for the raw key bytes we manage, no copy survives once the per-record keys are derived, and after you *lock* the vault a memory-dump scan finds zero copies.

**How to audit (after locking the vault, not while it's running).** First run the heap-scan unit tests for the raw key buffer: `pytest tests/unit/security/test_secure_key_buffer.py -v`. Then *lock* the vault and dump the Compute process — positional arguments, `<pid> <output.core>`: `sudo ./scripts/security/dump_process_memory.sh $(pgrep -f compute_service.py) compute.dump`. Scan the dump for the key bytes we control: `python scripts/security/scan_memory_for_key.py compute.dump --key-hex <your-known-key>`. Exit code 0 means zero copies — in both ASCII-hex and raw-binary form — of the controlled key bytes survive the lock.

## 04 Every persisted document field is encrypted, not just the obvious ones.

AT-REST AUDIT

**Contract.** Encrypting "the database" is not enough. Every metadata field that touches user content — file paths, query history, extracted text, OCR output, vector embeddings, timestamps — must run through the field-encryption layer. New ingestion formats fail this gate until they're wired through.

**How to audit.** Run the at-rest encryption test suites, which walk every persisted format and assert field-level encryption plus the entropy floor:

```
pytest tests/integration/test_encryption_at_rest.py \
  tests/integration/test_fts5_encryption_at_rest.py \
  tests/security/test_core_db_entropy.py -v
```

A green run means every persisted field — file paths, extracted text, OCR output, embeddings, and metadata — is encrypted at rest.

§3 — TEN QUESTIONS FOR ANY AI VENDOR

# The vendor checklist.

Use the list against any AI tooling you're considering — including SealedBrief. A vendor that can't answer all ten without an NDA, or that needs to ask its sales team, is a vendor whose security posture you can't audit. We've marked SealedBrief's answer next to each item.

**01** **Where do my documents go when I ask a question?**
A vendor that says "to our servers" is honest about being a cloud product. A vendor that says "depends on your subscription tier" is a vendor you can't audit. **SealedBrief: never leave your machine.** ✓

**02** **Can I monitor the network traffic and verify the previous answer?**
A vendor that can't be put under `tcpdump` fails the audit. **SealedBrief: see §2 claim 01.** ✓

**03** **Where does the encryption key live, and who can read it?**
A vendor that says "we manage encryption" controls the key. A vendor that says "your OS keychain" doesn't. **SealedBrief: OS keychain; the raw key bytes we control are pinned and wiped, and after you lock the vault a dump scan finds no copy (see §2 claim 03).** ✓

**04** **Is the cryptographic primitive named, audited, and standard?**
AES-256-GCM is the modern bar. Vendor-X-proprietary-cipher is not. **SealedBrief: AES-256-GCM for field-level encryption of document content and metadata (the vector-store text/metadata), plus SQLCipher (an audited library, AES-256 page encryption) for the index database.** ✓

**05** **What metadata gets stored, and where?**
Filenames, query history, embeddings — every field is a potential leak. A vendor that won't enumerate gets a no. **SealedBrief: §2 claim 04 enumerates the layer.** ✓

**06** **What logs the vendor keeps, and for how long?**
"We don't keep logs" is uncheckable. "We keep logs for 90 days at provider X" is. **SealedBrief: zero logs of document content; licence-validation access logs at sealedbrief.com retained 30 days; and if you explicitly opt in, crash reports go to Sentry, retained 90 days then purged.** ✓

**07** **Does the EULA permit human review of "data we collect"?**
"Authorised personnel may review submissions to improve our models" is a red flag. **SealedBrief: nothing collected, nothing to review.** ✓

**08** **What happens when the vendor goes out of business?**

A cloud product becomes inaccessible the day the API turns off. A locally-run product keeps working. **SealedBrief: the software keeps working offline with no server dependency after activation; the licence is a 3-year term, and when the term lapses the app opens your existing vault in read-only mode rather than going dark — your documents stay accessible. Renew to restore full function.** ✓

**09** **Can my IT team run a security review without an NDA?**

Vendors who require NDAs to discuss security rule out audits. **SealedBrief: architecture is published; claims are verifiable from the binary without an NDA.** ✓

**10** **Opt-in or opt-out for any data collection?**

Opt-in by default with explicit per-channel consent is the privacy-respectful baseline. Opt-out (or no toggle) means your data is the product. **SealedBrief: zero collection by default; opt-in crash-report channel only if you explicitly enable it.** ✓

§4 — CLOSING

# If this framework helps, we've succeeded.

Buy SealedBrief if your work involves documents that simply cannot ride a third-party API. Don't buy it if you need agentic workflow execution, cloud-scale inference across many users, or multi-user shared workspaces — the **What SealedBrief is not** section on the website enumerates what we don't do, in five explicit bullets, on purpose: filtering wrong-fit buyers before they buy is part of our design.

A 30-day no-questions refund is in place; statutory withdrawal and right-of-regret protections (where they apply to you) are unaffected by it. If SealedBrief doesn't fit your workflow, start the refund process at `sealedbrief.com/refund` or email **support@sealedbrief.com**. The licence is added to a revocation list on refund; documents you processed during the window remain yours.

## Contact

Technical questions about anything in this whitepaper: **support@sealedbrief.com** (or **security@sealedbrief.com** for security-specific reports).
Refund requests: **support@sealedbrief.com**.
Privacy / data-deletion requests: **privacy@sealedbrief.com**.
General support: **support@sealedbrief.com**.

Audit walkthroughs in operating-system terms ship with the product, and the audit scripts themselves live in the published source repository under `scripts/security/` alongside `docs/security/V1.0_LAUNCH_EVIDENCE.md`. SealedBrief v1.0 launches on Linux and macOS; Windows follows at V1.1.

---