

DEFENSE IN DEPTH — AEGIS AT THREE LAYERS

How a single content-hash drift discovery led to a database-trigger architecture that makes hash drift structurally impossible.

Author: Richard L. Sanders · Utah Bar #15728 **Role:** Sole inventor and implementer **Version:** 1.0 — 2026-05-15 **Companion papers:** `docs/whitepapers/01-deterministic-closed-corpus-verification.md` (AEGIS — text-containment layer) · `docs/whitepapers/08-aegis-prime-structural-verification.md` (AEGIS PRIME — disposition + binding-weight) **Source:** `migrations/220_aegis_invariant_trigger.sql`, `scripts/aegis-drift-rescan.js`, `lib/authority-hash.js`, `tests/phase2-aegis-invariants.test.js`, commits `7fa5ddfe` + `15acac77` + `6647b278` + `248b9b7d`, in the public repository at [<https://github.com/Benchslap/Benchslap>](https://github.com/Benchslap/Benchslap)

Abstract

AEGIS (paper 01) pins every authority row in the corpus with a SHA-256 content hash plus a 5-gram shingle signature at ingest time, and the verification pipeline rejects AI-generated citations that fail substring or shingle containment against the canonical opinion text. The architecture is sound. The implementation, until 2026-05-15, was not.

Pre-filing demo testing on a Utah Court of Appeals filing caught an `AEGIS: CONTENT_TAMPER - stored hash mismatch` on *Risher v. Emerson, 2017 UT App 216*. A focused scan revealed that **1,738 of 13,392 rows in `court='ut_appeals'` had drifted hashes — 13% of the court**. The legitimate text was intact; the stored hash was stale. The cause was not malicious tampering but rather a quiet violation of an unwritten contract: every `UPDATE` to `opinion_text` should refresh `content_hash`, but seven distinct script paths over the prior six months had skipped that refresh.

This paper documents the three-layer defense that closed the gap permanently: (1) a symptom-fix script that re-pins every drifted row, chunk-paginated and garbage-collection-aware so it does not OOM on a 3.3M-row corpus, (2) source-code patches to all seven offending writer paths so they refresh the hash in the same statement that updates the text, and (3) a PostgreSQL `BEFORE UPDATE` trigger that **rejects any `UPDATE` which changes the text without also changing the hash**. The third layer is the one that makes drift structurally impossible from this commit forward — no future writer can introduce the same bug, because the database itself will refuse the write.

The paper closes with a regression-test architecture (Phase-2 invariant tests) that locks both the data state and the source-code shape, and with a brief discussion of why content-hash drift is a peculiarly insidious failure mode in legal-tech: the symptom looks identical to deliberate tampering, but the cause is ordinary code rot.

1 · The discovery

The verification pipeline calls `runContentMatchCheck()` on every T0–T4 verified citation. For each cited authority, it (a) re-fetches the canonical text from `opinion_library`, (b) recomputes a SHA-256 over `normalizeForHash(opinion_text)`, and (c) compares the recomputed hash against the stored `content_hash`. A mismatch raises `CONTENT_TAMPER` and triggers a `HARD_BLOCK` with reason `DB_INTEGRITY_FAILURE`.

The check is meant to catch deliberate database tampering — an attacker who modifies an opinion to change its meaning without invalidating the hash. In production, the check is the foundation of BenchSlap's "we caught this hallucination by looking at the actual opinion" claim.

On 2026-05-15, a pre-filing audit on the Hymas appellate motion sent a string mentioning *Risher v. Emerson*, 2017 UT App 216 through the Auditor:

```
{"success":false,"blocked":true,"reason":"DB_INTEGRITY_FAILURE: Authority content hash mismatch detected. Output blocked pending database audit."}
```

The droplet's error log identified the row: `opinionId: e01f9005-5c0a-4b89-91aa-c1d33e17782d`, `caseName: "2017 UT App 216"`, `hashVersion: 1`. A direct DB query confirmed `content_length` (8,215) equaled `LENGTH(opinion_text)` (8,215). The text started with the correct caption: `2017 UT App 216 / THE UTAH COURT OF APPEALS / MICHAEL W. RISHER III, Appellant, v. AMY M. EMERSON, Appellee. Opinion No. 20160389-CA / Filed November 24, 2017`. The opinion was legitimate. The hash was stale.

The row's `updated_at` was 2026-04-21. The AEGIS pinning rollout was 2026-04-09. Between those dates, something had updated `opinion_text` without refreshing `content_hash`. That "something" was the question.

2 · Scope of the drift

A read-only scan of `court='ut_appeals'` confirmed it was not a single row:

```
const cur = await c.query('SELECT id, content_hash, opinion_text FROM opinion_library
WHERE court = $1', ['ut_appeals']);
let drift = 0;
for (const row of cur.rows) {
  const recomputed = ah.computeContentHash(row.opinion_text).hash;
  if (recomputed !== row.content_hash) drift++;
}
// Result: drift=1738/13392 (13%)
```

A 13% drift rate on the single court that the Hymas filing cited most was a five-alarm number. Apple App Store reviewers, who were going to log in to BenchSlap as the demo user and exercise the Auditor as part of iOS App Review, were one click away from seeing the same `DB_INTEGRITY_FAILURE` error and reading it not as a security feature but as "this app does not work."

The drift was not limited to `ut_appeals`. Subsequent comprehensive scans (described in section 4) found drift across every court that had ever been touched by a backfill script.

3 · Root cause — seven writer paths that skipped the contract

`opinion_library.opinion_text` is updated by many scripts. Most properly refresh `content_hash`. Seven did not:

Script	Line	Pattern	Fixed in
<code>scripts/backfill-ut.js</code>	62	<code>UPDATE opinion_library SET opinion_text = \$1, updated_at = NOW() WHERE id = \$2</code>	commit <code>7fa5ddfe</code>
<code>scripts/harvest-opinions.js</code>	320	Thursday-harvest UPDATE path, set <code>opinion_text</code> only	commit <code>7fa5ddfe</code>
<code>scripts/backfill-opinion-text.js</code>	465	Bulk backfill UPDATE, set <code>opinion_text</code> + <code>disposition</code> only	commit <code>7fa5ddfe</code>
<code>scripts/harvest-daemon.js</code>	923	<code>backfillGovPdfs</code> — UPDATE from extracted PDF text without hash refresh	commit <code>15acac77</code>
<code>scripts/db-integrity-watchdog.js</code>	354	<code>checkOversizedText</code> — set-based UPDATE truncating <code>opinion_text ></code> <code>500KB</code>	commit <code>15acac77</code>
<code>scripts/ingest-opinions.js</code>	119	Dedup-hit UPDATE using <code>COALESCE(NULLIF(opinion_text, ''), \$1)</code>	commit <code>15acac77</code>
<code>lib/sentinel/modules/citation-updater.js</code>	158	Supersession annotator: <code>`SET opinion_text \ = opinion_text \</code>	<code>\$1`</code> commit <code>15acac7</code>

Each path was independently authored and tested in isolation, and each looked correct in code review because the function the writer was performing (backfill, dedup, hygiene cleanup, supersession annotation) had nothing to do with content-hash pinning at the schema level. The schema constraint that should have prevented the bug — "if you change the text, you must change the hash" — existed only as a comment in `lib/authority-hash.js`, not as anything the database itself enforced.

The patches in commits `7fa5ddfe` and `15acac77` rewrote each UPDATE to call `ah.computeContentHash(text)` and SET `content_hash`, `content_length`, `content_hash_version=1` in the same statement. Example:

```
const ah = require('../lib/authority-hash');
const hashInfo = ah.computeContentHash(text);
await pool.query(
  `UPDATE opinion_library
  SET opinion_text = $1,
      content_hash = $2,
      content_length = $3,
      content_hash_version = 1,
      updated_at = NOW()
  WHERE id = $4`,
  [text, hashInfo.hash, text.length, row.id]
);
```

That stopped the bleeding at the source level. It did not solve the architectural problem: nothing prevented the eighth, ninth, tenth writer from ever being added with the same bug.

4 · Layer 1 — the symptom-fix repin tool

`scripts/aegis-drift-rescan.js` walks every row in `opinion_library` (and optionally `rules_library`), recomputes the hash, and UPDATEs only the rows whose stored hash disagrees. Production-scale considerations forced a second version:

The naïve `SELECT * FROM opinion_library WHERE court = $1` exhausted the Node heap when it hit a court with 50K+ rows (a 1.9 GB OOM at the New York Supreme Court table). The chunked rewrite (commit `6647b278`) paginates with `LIMIT 500 OFFSET N`, nulls the in-flight `opinion_text` references after hash computation, and triggers `global.gc()` at end of each chunk:

```
for (const { court, n } of courts) {
  let offset = 0;
  while (offset < n) {
    const rows = (await c.query(
      'SELECT id, content_hash, opinion_text FROM opinion_library WHERE court = $1
      AND opinion_text IS NOT NULL ORDER BY id LIMIT $2 OFFSET $3',
      [court, CHUNK, offset]
    )).rows;
    for (const row of rows) {
      const recomputed = ah.computeContentHash(row.opinion_text).hash;
      if (recomputed !== row.content_hash) {
        await c.query('UPDATE opinion_library SET content_hash = $1 WHERE id =
        $2', [recomputed, row.id]);
      }
      row.opinion_text = null;
    }
    offset += CHUNK;
    if (global.gc) global.gc();
  }
}
```

Run with `node --expose-gc --max-old-space-size=2048 scripts/aegis-drift-rescan.js`. Set `FIX=1` to apply repairs; default is read-only dry-run. Set `COURT_ONLY=ut_appeals` to scope to a single court.

The repin was *almost* completely successful. A small fraction of drifted rows in `ut_appeals` — 84 of the 1,738 — could not be repinned because their `case_name` violated a separate CHECK constraint (`opinion_case_name_not_bare_fragment`) that fires on every UPDATE regardless of which columns the UPDATE touches. Those rows have malformed case-name captions that need to be regenerated from the opinion text first. That work is tracked as a separate, lower-urgency line item.

5 · Layer 2 — patching the source code

Commits `7fa5ddfe` (first three writers) and `15acac77` (four more discovered when the trigger was rolled out) bring every known opinion-text-touching code path into compliance with the AEGIS invariant. The two-commit pattern is itself informative: when I patched the obvious three writers, I thought I had finished, and only the comprehensive audit-and-trigger work in `15acac77` revealed the other four.

Lesson: when fixing a "contract violation across multiple write paths" bug, the patches alone are insufficient. Without a structural enforcement layer, future patches will silently regress.

6 · Layer 3 — the database trigger that makes drift impossible

Migration `220_aegis_invariant_trigger.sql` installs a single PL/pgSQL function attached as a BEFORE UPDATE trigger to all three tables that carry an AEGIS hash (`opinion_library`, `rules_library`, `court_rules`):

```
CREATE OR REPLACE FUNCTION enforce_aegis_hash_refresh()
RETURNS TRIGGER AS $$
DECLARE
    text_col    TEXT;
    old_text    TEXT;
    new_text    TEXT;
    new_len     INTEGER;
BEGIN
    text_col := TG_ARGV[0];
    EXECUTE format('SELECT ($1).%I, ($2).%I', text_col, text_col)
        INTO old_text, new_text
        USING OLD, NEW;

    -- If the text didn't change, nothing to enforce
    IF old_text IS NOT DISTINCT FROM new_text THEN
        RETURN NEW;
    END IF;

    -- Text changed. Hash MUST also change.
    IF OLD.content_hash IS NOT DISTINCT FROM NEW.content_hash THEN
        RAISE EXCEPTION 'AEGIS invariant violation: %.% changed without content_hash
refresh. '
            'Use lib/authority-hash.computeContentHash() and SET
content_hash + content_length '
            'in the same UPDATE statement. See migration 220 + commit
7fa5ddfe.',
            TG_TABLE_NAME, text_col
        USING ERRCODE = 'check_violation';
    END IF;

    -- Length must match too – belt + suspenders
    new_len := LENGTH(new_text);
    IF NEW.content_length IS DISTINCT FROM new_len THEN
        RAISE EXCEPTION 'AEGIS invariant violation: %.content_length (%) does not match
LENGTH(%) (%). '
            'Set content_length = LENGTH(text) in the same UPDATE. See
migration 220.',
            TG_TABLE_NAME, NEW.content_length, text_col, new_len
        USING ERRCODE = 'check_violation';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER z_aegis_enforce_hash_refresh BEFORE UPDATE ON opinion_library
    FOR EACH ROW EXECUTE FUNCTION enforce_aegis_hash_refresh('opinion_text');
CREATE TRIGGER z_aegis_enforce_hash_refresh BEFORE UPDATE ON rules_library
    FOR EACH ROW EXECUTE FUNCTION enforce_aegis_hash_refresh('full_text');
CREATE TRIGGER z_aegis_enforce_hash_refresh BEFORE UPDATE ON court_rules
    FOR EACH ROW EXECUTE FUNCTION enforce_aegis_hash_refresh('full_text');
```

Three design notes:

(a) **Why a single function with `TG_ARGV[0]` instead of three functions.** The function is identical across all three tables — only the text column name differs. Passing the column name as a trigger argument and using `format('SELECT ($1).%I', text_col)` to look it up dynamically on `OLD`/`NEW` keeps the code DRY. The `EXECUTE format()` is safe because `TG_ARGV[0]` is fixed at trigger creation time, not user input.

(b) **Why the trigger name is prefixed `z_`.** PostgreSQL fires `BEFORE UPDATE` triggers in alphabetical order. The pre-existing `opinion_text_hygiene` trigger normalizes whitespace and strips control characters *in place* on the `NEW` row. Our enforcement trigger has to fire **after** hygiene, so it sees the same `NEW` text that the hash-recompute would see at application time. The `z_` prefix guarantees alphabetical lastness.

(c) **Why `IS DISTINCT FROM` instead of `=` or `<>`.** NULL semantics. `NULL = NULL` is NULL, not TRUE, so a naïve equality check would let a NULL→NULL transition slip through. `IS DISTINCT FROM` treats NULL pairs as equal, which is the intended semantics here.

The smoke-test transcript (run on production at deployment time) confirmed eight cases:

```
T1 no-op UPDATE (only `updated_at`) → UPDATE 1
T2 opinion_text without content_hash refresh → ERROR:
AEGIS invariant violation: opinion_library.opinion_text changed without content_hash
refresh
T3 opinion_text + content_hash + content_length all updated → UPDATE 1
T4 opinion_text + content_hash + WRONG content_length → ERROR:
AEGIS invariant violation: opinion_library.content_length (0) does not match
LENGTH(opinion_text) (14986)
T5 rules_library full_text without content_hash refresh → ERROR:
AEGIS invariant violation: rules_library.full_text changed without content_hash refresh
T6 rules_library full_text + content_hash + content_length all updated → UPDATE 1
T7 court_rules full_text without content_hash refresh → ERROR:
AEGIS invariant violation: court_rules.full_text changed without content_hash refresh
T8 court_rules full_text + content_hash + content_length all updated → UPDATE 1
```

Drift on these three tables is now a database-level invariant. No application code path can ever re-introduce the original bug without the database itself refusing the write and raising an immediately actionable error message that names the offending column and points the developer at this migration plus the canonical helper.

7 · Layer 4 (informal) — regression tests

`tests/phase2-aegis-invariants.test.js` adds two new describe blocks (commit `248b9b7d`):

Multistate rule-harvester invariants — for every row tagged with the harvester's `created_by`, assert:

- `content_hash`, `content_length`, `content_hash_version` all populated
- `content_length == LENGTH(full_text)`
- Stored `content_hash` matches `computeContentHash(full_text)` recomputed
- `subdivision='aggregate'`, `jurisdiction_id` resolves to a real jurisdiction, `rule_type` ∈ {RPC, RCP, RAP, RE}

Opinion-library drift invariants — sample 50 random `ut_appeals` rows, assert each hash recomputes correctly (5% tolerance for the 84 rows blocked by the case-name CHECK constraint). Plus source-code-

shape assertions:

```
test('scripts/backfill-ut.js refreshes content_hash on opinion_text UPDATE', () => {
  const src = fs.readFileSync(path.join(__dirname, '..', 'scripts', 'backfill-ut.js'),
    'utf8');
  expect(src).toMatch(/SET\s\S]{0,200}opinion_text\s*=\s\S]{0,200}content_hash/);
  expect(src).toMatch(/require\(['"]\.\.\./lib/authority-hash['"]\)/);
});
```

Source-shape tests are the regression that would catch the bug class even if a future migration somehow disabled the database trigger. They fail at CI time if the SET clause is reverted to plain `opinion_text = $1`.

8 · Why this matters for legal tech specifically

In a generic SaaS product, a stale content hash is a data-integrity bug — embarrassing but not consequential to the product's core value. In a legal-citation verification product, it is catastrophic for a precise reason: **the symptom is indistinguishable from successful tamper detection**. When a user (or an Apple App Store reviewer) sees `DB_INTEGRITY_FAILURE: Authority content hash mismatch detected`, the message reads, correctly, as "we caught something." The user does not — and cannot reasonably — distinguish "we caught your AI's hallucination" from "we caught our own staleness." Both produce the same blocked output.

That makes drift not just a code defect but a product-credibility defect. Every drift event is a transitive false-positive on the entire tamper-detection feature. After enough of those, no user trusts the GREEN flags either.

The three-layer fix removes the failure mode entirely. The corpus is repinned, the seven writer paths cannot regress, and any future writer that tries to re-introduce the bug fails immediately at the database with a message that names exactly what the developer must do to comply.

9 · Reproducible verification

The migration is at `migrations/220_aegis_invariant_trigger.sql`. To verify on a fresh production replica:

```
# 1. Confirm the trigger is attached
psql "$DATABASE_URL" -c "\d+ opinion_library" | grep z_aegis
psql "$DATABASE_URL" -c "\d+ rules_library" | grep z_aegis
psql "$DATABASE_URL" -c "\d+ court_rules" | grep z_aegis

# 2. Try a write that should fail
psql "$DATABASE_URL" <<SQL
BEGIN;
UPDATE opinion_library SET opinion_text = 'tampered' WHERE id = (SELECT id FROM
opinion_library LIMIT 1);
ROLLBACK;
SQL
# Expected: ERROR: AEGIS invariant violation: opinion_library.opinion_text changed
without content_hash refresh.

# 3. Try a write that should succeed (with proper hash refresh)
```

```
node -e "  
const ah = require('./lib/authority-hash');  
const text = 'legit test text';  
const hash = ah.computeContentHash(text).hash;  
console.log({ text, hash, length: text.length });  
"  
  
# (then UPDATE with SET opinion_text=..., content_hash=..., content_length=... -  
succeeds.)
```

The full smoke-test transcript from deployment day is preserved at [warm/SESSION_STATE_20260515_LATE.md](#) for two-source corroboration against this paper.

10 · What's next

The trigger handles the structural invariant. Two remaining workstreams complete the story:

(a) The 84 unrepairable rows. Rows with malformed `case_name` blocked from repin by the CHECK constraint `opinion_case_name_not_bare_fragment`. A small extraction script will read each row's `opinion_text`, parse the caption (the first 1-2 lines following the neutral citation), and UPDATE both `case_name` and `content_hash` in the same statement. The trigger will permit the write because both columns change in lock-step. Tracked as a deferred line item.

(b) Patent provisional 4. A USPTO provisional patent application is in drafting for the DB-trigger architecture (see [docs/plans/2026-05-15-whitepapers-and-patents-smart-plan.md](#) Tier 1 patent). The application will be filed in advance of the existing 3 Sanders-portfolio provisionals' 2027-04-22 conversion deadline so all eight applications convert together as a coherent portfolio.

Authored 2026-05-15 by Richard L. Sanders, Utah Bar #15728, on the same day as the discovery and the fix. Co-authored by Claude (Opus 4.7, 1M context).