FaultKey CausalLayer

Engine Technical Brief

Every capability of the deterministic AI liability attribution engine, in one document.

Engine v0.5.0 · Anchor schema v1 · Ledger schema v1 · 17 May 2026

Author: FaultKey · Brisbane, Australia · hello@faultkey.com · https://faultkey.com

---

Executive Summary

FaultKey CausalLayer is a **deterministic, closed-form, MCP-exposed AI liability attribution engine**. Given a structured incident description — agents, events, severity, jurisdiction, financial impact — the engine produces a **CausalCertificateV1**: a signed, hash-chained, Bitcoin-anchored receipt that allocates fault between the AI vendor, the deployer, and the end-user, quantifies damages, maps regulatory exposure, and emits an underwriting recommendation.

There is **no LLM in the attribution path**. The same input produces byte-identical output every call. End-to-end wall time is under 300 ms for the standard pipeline and approximately 2 ms per perturbation for the counterfactual simulator. The engine ships as 147,051 lines of strict-mode TypeScript across 182 modules, a four-layer cryptographic ledger, and an open Model Context Protocol (MCP) reference server.

The pipeline runs sixteen analysis modules in a single deterministic pass: causal graph construction, deviation taxonomy, liability mode analysis, four-factor liability scoring, cross-case calibration, three-layer attribution, foreseeability, counterfactual simulation, contributory negligence, regulatory mapping, precedent matching, damages quantification, insurance underwriting, actuarial pricing, financial blast radius, and pre-deployment stress testing. Outputs are cryptographically committed to a hash-chained JSONL ledger, summarised in a daily Merkle root, signed with Ed25519, anchored to Bitcoin via

OpenTimestamps, and mirrored to a public anchor-log Git repository so any third party can verify any past attribution without trusting FaultKey.

The engine is mapped in detail to **APRA CPS 230**, **EU AI Act Articles 9, 12, 13, 14, 26, 50**, **NIST AI Risk Management Framework**, **ISO/IEC 42001**, and Australia's **Voluntary AI Safety Standard**. Production tenants receive real Ed25519 signatures, a managed OpenTimestamps anchor, ACORD claims-export integration, and an SLA. The free public demo at `https://mcp.faultkey.com/mcp` issues unsigned exemplar certificates that demonstrate the full engine surface without exposing the proprietary scoring constants.

This document is the single canonical reference for what the engine does, how it does it, what it commits to cryptographically, what it does not do, and how a third party can verify its outputs independently.

Capability Map

The engine exposes capabilities along seven layers. Each layer is independently auditable and depends only on the layers beneath it.

#	Layer	Module count	Purpose
7	Output and export	12	Report generation, ACORD claims export, EU AI Act Article 73 incident report, evidence packaging, MCP envelope, REST envelope
6	Insurance and actuarial	8	Underwriting score, actuarial pricing, portfolio aggregation, reinsurance triggers
5	Financial calibration	14	Damages quantification, financial ranges, Platt calibration curves, outcome-feedback ingestion
4	Legal analysis	18	Foreseeability, contributory negligence, vicarious liability, litigation risk, court attribution
3	Liability attribution	12	Four-factor weighted scoring, exact Shapley values, three-layer decomposition, counterfactual reasoning
2	Causal and regulatory	16	Causal graph construction, regulatory mapping (APRA, EU AI Act, NIST, ISO 42001, sectoral), deviation taxonomy, authority-boundary gaps
1	Data and precedent	22	AIID ingestion, precedent matching, case-law corpus, historical case alignment
0	Self-improvement	10	Cross-case learning, outcome calibration, empirical weight calibration, flywheel ingestion
—	Supporting	56	Adversarial test rigs, blind benchmarks, monitoring, SDK, validation harnesses

The most-depended-upon modules, ranked by import count inside the engine, are `causalGraph.ts` (90), `liabilityScorer.ts` (84), `regulatoryMapper.ts` (22), `boundaryGapDetector.ts` (14), `damagesQuantification.ts` (12), `counterfactualSimulator.ts` (12), `precedentMatcher.ts` (11), and `insuranceUnderwriting.ts` (10).

The Sixteen-Module Pipeline

A single `submit_incident` call fans out through the modules below in deterministic sequence. Every module is a pure function of its inputs; intermediate state is hashed and recorded in the ledger entry alongside the final certificate.

1. Causal graph construction

Builds a directed acyclic graph from the timestamped event sequence. Identifies the root cause, all propagation paths, and the integrity of the causal chain. Applies the **but-for** standard of causation: would the harm have occurred but for this agent's action.

2. Deviation taxonomy classification

Classifies each agent deviation against a canonical taxonomy of seventeen AI failure modes including specification gaming, reward hacking, distributional shift, capability overhang, goal misgeneralisation, prompt injection, jailbreak, data poisoning, sensor occlusion, and ODS non-compliance. Returns a coverage percentage and per-deviation confidence. As of v0.5 the abstain rate is 0 percent.

3. Liability mode analysis

Determines the legal theory of liability applicable to the incident. Categories are **product defect**, **operational failure**, **hybrid**, and **strict liability** (for physical-harm safety-critical systems such as medical devices and autonomous vehicles). Eleven signal detectors evaluate agent autonomy levels, oversight patterns, sector characteristics, and narrative indicators.

4. Four-factor liability scoring

Each agent receives a weighted score on four dimensions: **causal proximity** (30 percent), **behavioural deviation** (30 percent), **controllability** (20 percent), and **regulatory alignment** (20 percent). Per-agent percentages sum to 100 percent. Cross-case

calibration in module 5 then adjusts these raw weights category-by-category and jurisdiction-by-jurisdiction.

5. Cross-case calibration

Adjusts raw liability scores using learned patterns from 725 resolved outcomes (209 verified, 516 inferred). Calibration is the blend of three signals — category-specific weight profiles (27 categories with detected biases, weight 50 percent), jurisdiction-specific Platt sigmoid curves (six jurisdictions reliably calibrated, weight 30 percent), and cross-case correlations (37 actionable patterns, weight 20 percent) — capped at plus-or-minus eight percentage points to prevent over-correction.

6. Three-layer liability attribution

Maps fault across the three legal layers required by most jurisdictions: **direct liability** (the agent that caused the harm), **vicarious liability** (the operator or deployer responsible for the agent), and **contributory negligence** (third parties whose actions contributed). Applies the doctrines of non-delegable duty, joint enterprise, and respondeat superior.

7. Foreseeability analysis

Assesses whether the harm was reasonably foreseeable at the time of deployment. Considers prior incidents in the same sector, state of the art, published research, regulatory warnings, and the deployer’s own documented internal risk assessments.

8. Counterfactual simulator

For every incident the engine automatically runs eighteen counterfactual perturbations such as “if the human operator had acted instantly,” “if the guardrail had been active,” “if the retrieval index had been current,” and outputs a mathematical proof of but-for causation in the language used in subrogation: *Even if the human operator had acted correctly, the AI Provider’s liability would have changed by less than five percentage points — supporting but-for causation against the AI Provider.* Wall time is approximately two milliseconds per perturbation.

9. Contributory negligence assessment

Evaluates whether the claimant contributed to their own harm through failure to read warnings, misuse, assumption of known risk, or failure to mitigate.

10. Regulatory compliance mapping

Maps the incident against applicable frameworks: EU AI Act Articles 6, 9, 14, 15, and 50 with penalty exposure calculation; UK AI Safety Framework; United States state-level AI legislation across 78 bills in 27 states; sectoral regulators including the United States Food and Drug Administration, National Highway Traffic Safety Administration, Financial Conduct Authority, Australian Prudential Regulation Authority, Therapeutic Goods Administration, Australian Securities and Investments Commission; and ISO/IEC 42001 management-system controls.

11. Precedent matching

Searches a corpus of resolved AI liability cases for the closest analogues. Returns similarity scores, outcome summaries, and applicable legal principles from prior decisions.

12. Damages quantification

Calculates expected financial exposure across direct damages, consequential damages, and punitive damages where jurisdiction caps allow. Calibrated against fourteen resolved cases with an R-squared regression fit of 0.86.

13. Insurance underwriting

Produces a full underwriting assessment: risk score (0–1000), risk grade (A–E), recommendation (`ACCEPT`, `ACCEPT_WITH_CONDITIONS`, `DECLINE`), expected annual loss calibrated to 1.73 times the stated financial impact, technical premium calculation, and policy exclusions.

14. Actuarial pricing

Calculates the insurance premium using a base rate from the risk score, a sector loading factor, a claims-history adjustment, and a jurisdiction multiplier. Produces gross premium, net premium, and a loss-ratio estimate.

15. Financial blast radius

Models total financial exposure per agent including direct liability allocation, defence costs (18 percent loading on large claims), regulatory fines, reputational damage multiplier, and business-interruption costs.

16. Pre-deployment stress test

Runs eight failure scenarios against the system architecture: adversarial input injection, cascading multi-agent failure, data poisoning, model drift under distribution shift, oversight-mechanism bypass, authority-boundary violation, regulatory-change exposure, and supply-chain dependency failure. Returns per-scenario risk scores and a `PASS` or `DENY` certification verdict.

Mathematical Methods

Four-factor weighted scoring

For each agent a in incident i, the raw liability share L_a^{raw} is

$$L_a^{\text{raw}} = w_p \cdot P_a + w_d \cdot D_a + w_c \cdot C_a + w_r \cdot R_a$$

with $w_p = 0.30$ (causal proximity), $w_d = 0.30$ (behavioural deviation), $w_c = 0.20$ (controllability), and $w_r = 0.20$ (regulatory alignment). Per-agent shares are then normalised so the vector sums to unity, then passed to cross-case calibration.

Exact Shapley enumeration

For multi-agent incidents the engine enumerates the exact Shapley value for each player using the standard formulation:

$$\phi_a(v) = \sum_{S \subseteq N \setminus \{a\}} \frac{|S|! \cdot (|N| - |S| - 1)!}{|N|!} \cdot \left(v(S \cup \{a\}) - v(S) \right)$$

where $v(S)$ is the harm-attribution function evaluated on coalition S. This is computed exactly (not approximated) because the engine deliberately caps the player set at small cardinality; the resulting allocation satisfies the four classical Shapley axioms — efficiency, symmetry, dummy, and additivity — which is what makes the certificate defensible in court rather than merely plausible.

Platt calibration

Per-jurisdiction confidence is calibrated using a sigmoid fit:

$$P(y = 1 \mid s) = \frac{1}{1 + \exp(A \cdot s + B)}$$

with A and B fit by maximum likelihood on the resolved-outcome set for each jurisdiction. Six jurisdictions currently meet the minimum-30-outcomes threshold for reliability (United States, United Kingdom, European Union, Australia, Canada, Singapore).

Counterfactual but-for proof

For each candidate counterfactual perturbation π on the input x, the engine recomputes the full pipeline on $\pi(x)$ and records the maximum change in primary-party share. If $\max_{\pi} \left| L_{\text{primary}}(x) - L_{\text{primary}}(\pi(x)) \right| < \tau$ for $\tau = 0.05$, the engine certifies but-for causation against the primary party at the five-percentage-point threshold.

CausalCertificateV1 — Output Schema

Every successful `submit_incident` call returns a `CausalCertificateV1`. The structure below is the canonical, deterministic JSON shape; the same input always produces the

same bytes.

```
{
  "schemaVersion": 1,
  "certificateId": "<sha256(inputHash || outputHash || timestamp)>",
  "issuedAt":      "2026-05-17T01:23:45.678Z",
  "engineVersion": "0.5.0",
  "incident": {
    "title":               "<string>",
    "severity":            "low|medium|high|critical",
    "jurisdiction":        "AU|US|EU|UK|CA|SG|...",
    "financialImpactCents": 420000,
    "deterministicOnly":    true
  },
  "verdict": {
    "kind":
"single_party|shared_liability_two_party|shared_liability_three_party",
    "primaryParty":    "<agentId>",
    "primaryShare":    0.594,
    "secondary":       [
      { "party": "<agentId>", "share": 0.203 },
      { "party": "<agentId>", "share": 0.203 }
    ],
    "confidence":      0.72,
    "calibratedBy":    "725 resolved outcomes"
  },
  "causalGraph": {
    "nodes": [...],
    "edges": [...],
    "rootCause": "<nodeId>",
    "butForChain": ["<nodeId>", "<nodeId>", "<nodeId>"]
  },
  "deviationTaxonomy": [...],
  "liabilityMode":
"product_defect|operational_failure|hybrid|strict_liability",
  "threeLayerAttribution": {
    "direct":      [...],
    "vicarious":   [...],
    "contributory":[...]
  },
  "foreseeability": { "score": 0.83, "evidence": [...] },
  "counterfactuals": {
    "perturbationsRun": 18,
    "maxSwingPP":        0.041,
    "butForProven":      true,
    "primaryParty":      "<agentId>"
  },
  "regulatorRelevant": {
```

```
    "EU_AI_Act_Art_9":   { "applies": true,  "violation": false },
    "EU_AI_Act_Art_12":  { "applies": true,  "violation": true,  "evidence":
"..." },
    "EU_AI_Act_Art_26":  { "applies": true,  "violation": true,
"penaltyExposure": "EUR 15M" },
    "APRA_CPS_230":      { "applies": true,  "violation": true,  "control":
"Operational risk" },
    "NIST_AI_RMF":       { "applies": true,  "function": "Manage",
"subcategory": "MG-3.2" },
    "ISO_IEC_42001":     { "applies": true,  "control": "A.6.1.4" },
    "AU_VAISS":          { "applies": true,  "principle": "Accountability" }
  },
  "precedents": [...],
  "damages": {
    "directCents":         850000,
    "consequentialCents":  650000,
    "punitiveCents":       0,
    "totalCents":          1500000,
    "rangeLowCents":       1200000,
    "rangeHighCents":      2100000,
    "calibrationR2":       0.86
  },
  "underwriting": {
    "riskScore":            720,
    "grade":                "C",
    "recommendation":       "ACCEPT_WITH_CONDITIONS",
    "expectedAnnualLossCents": 1800000,
    "exclusions":           [...],
    "conditions":           [...]
  },
  "actuarial": {
    "grossPremiumCents":    105000,
    "netPremiumCents":       89000,
    "lossRatioEstimate":     0.55
  },
  "blastRadius": {
    "perAgent":             [...],
    "totalExposureCents":   2640000
  },
  "stressTest": {
    "scenariosTested":      8,
    "certification":        "CONDITIONAL_PASS",
    "highestRisk":          "adversarial_input_injection"
  },
  "discoverySubpoenaChecklist": [
    { "ask": "bias audit log",  "ifFoundMaxSwingPP": 0.12, "priority": 1 },
```

```
    { "ask": "guardrail config","ifFoundMaxSwingPP": 0.08, "priority": 2 }
  ],
  "anchor": {
    "decisionId":    "<sha256>",
    "prevHash":      "<sha256>",
    "merkleRoot":    "<sha256>",
    "ed25519PubKeyFingerprint":
"5b7fc9b398b162e4900f43bddf55cda93c8c7d0b1749cc86e0cbb5754582d6e6",
    "ed25519Signature":         "<base64>",
    "openTimestampsProof":      "<base64-ots>",
    "bitcoinBlockHeight":       872341,
    "bitcoinBlockHash":         "<hex>",
    "anchorLogRepo":            "https://github.com/smq9sn5jck-
coder/causallayer-anchor-log",
    "status":                   "production|pre-genesis-test|demo_ephemeral"
  },
  "timing": { "totalMs": 201, "perturbationsMs": 36 }
}
```

Every field marked with `<sha256>` or `<base64>` is computed over the canonical-JSON serialisation of the relevant input or sub-object so a third-party verifier with only the certificate JSON can reconstruct each hash and confirm the chain without contacting FaultKey.

The Anchored Decision Ledger

A static percentage attribution is an opinion. An attribution that was provably generated at a specific moment in time, by a specific engine version, on a specific input hash, and immediately committed to a tamper-evident ledger that is later witnessed by the Bitcoin block chain, is a **legal instrument**. The Anchored Decision Ledger is what turns CausalLayer from a calculator into a forensic record-keeper.

It has four cryptographic layers, each strictly weaker-or-equal in trust assumption to the layer beneath it. An attacker has to break **all four simultaneously** to forge a back-dated attribution.

Layer	Trust required	Failure mode if attacked
1. JSONL ledger entry	The operator’s filesystem integrity for the current day	Detected the moment the next anchor is built
2. Daily Merkle root, Ed25519-signed	The operator’s signing key	Detected by anyone who saved a previous signature and ran `verify-anchor.js`
3. OpenTimestamps Bitcoin attestation	SHA-256 plus the existence of Bitcoin’s block headers	Requires reorganising Bitcoin
4. Public anchor-log Git repository	GitHub *or* any third-party mirror of the repo	Detected by any divergence between mirrors

Layer 1 — Hash-chained JSONL ledger

`server/engine/decisionLedger.ts` is an append-only JSONL writer. Every entry is canonical-JSON-serialised and hash-chained:

```
{
  "schemaVersion": 1,
  "decisionId":   "<sha256(inputHash || outputHash || timestamp)>",
  "prevHash":     "<previous entry's decisionId, or '' for the first>",
  "timestamp":    "2026-05-15T12:34:56.789Z",
  "engineVersion": "0.5.0",
  "moduleVersions": { "M1_BiasAnalyzer": "1.0.0", ... },
  "environment":  "production",
  "inputHash":    "<sha256 of canonical-JSON(input)>",
  "outputHash":   "<sha256 of canonical-JSON(output)>",
  "inputBytes":   1234,
  "outputBytes":  4567
}
```

The `prevHash` field links every entry to the previous one, which means a single tampered byte invalidates every subsequent decision and is detectable by any third party who has saved one later signature.

Layer 2 — Daily Merkle root, signed Ed25519

At the close of each UTC day the engine builds a Merkle tree over every `decisionId` produced that day, signs the root with the operator's Ed25519 key, and writes both root and signature to the daily anchor file. Current canonical key fingerprint:

> `5b7fc9b398b162e4900f43bddf55cda93c8c7d0b1749cc86e0cbb5754582d6e6`

Layer 3 — OpenTimestamps Bitcoin attestation

The signed Merkle root is timestamped via OpenTimestamps, which aggregates it into a calendar tree and waits for Bitcoin block inclusion. Once a Bitcoin block confirms the root, the resulting `.ots` proof is self-validating against any Bitcoin full node forever — without any reliance on FaultKey, OpenTimestamps, or any other party remaining online.

Layer 4 — Public anchor-log repository

Every signed daily root, every `.ots` proof, and every public-key rotation is published to the public, append-only Git repository `smq9sn5jck-coder/causallayer-anchor-log`. Any party can clone the repo and run the verifier to independently confirm any past attribution.

Pre-genesis notice

All anchors produced before the public genesis block are signed with a status field of `pre-genesis-test`, in line with the public commitment in `causallayer-anchor-log/PRE-GENESIS-NOTICE-2026-05-15.md`, so historical readers can never confuse a developer-mode anchor for a production one.

MCP Tool Surface

The engine is exposed over the Model Context Protocol (Streamable HTTP transport, protocol version `2024-11-05`) at `https://mcp.faultkey.com/mcp`. Four tools are

published by `tools/list`:

Tool	Purpose
`submit_incident`	Run the full sixteen-module pipeline and return a `CausalCertificateV1`.
`verify_certificate`	Verify a previously issued certificate against the public key, the OpenTimestamps proof, and the public anchor-log.
`get_anchor_status`	Return the current anchor-log head, latest Merkle root, latest Bitcoin block reference, and key-rotation history.
`query_issuer_registry`	Return the engine's signing-key history, version table, and module versions for any past `issuedAt`.

A minimal end-to-end client uses three calls: `initialize` (handshake, returns `Mcp-Session-Id`), `notifications/initialized` (state transition), and `tools/call` `submit_incident` with the agents and events arrays. Schema validation is strict: `agents[]` requires `id`, `name`, and `type`; `events[]` requires `id`, `type`, `timestamp`, and `description`. Malformed inputs receive a clean Zod error so any agent client can self-correct.

REST mirror

The same engine is reachable over plain HTTPS for non-agent callers:

```
POST https://api.faultkey.com/v1/incidents/analyze
Authorization: Bearer cl_live_xxxx
Content-Type: application/json
```

The request body and response schema match the MCP `submit_incident` arguments and `CausalCertificateV1` exactly, byte for byte.

Standards Mapping (Production Tenant)

Standard	Articles / Controls	What the certificate provides
EU AI Act	Articles 9 (risk management), 12 (record keeping), 13 (transparency), 14 (human oversight), 26 (deployer obligations), 50 (transparency for limited-risk), 73 (serious-incident reporting)	A timestamped, signed, Bitcoin-anchored event log per Article 12; an Article 73 incident report; a deployer-side record per Article 26; penalty-exposure calculation per article.
APRA CPS 230	Operational risk management, third-party reliance, business continuity	A deterministic operational-risk attribution between the regulated entity and its AI-vendor third party, with a tamper-evident ledger entry each time a regulated decision is made.
NIST AI RMF	Map / Measure / Manage / Govern functions and sub-categories	Per-incident mapping of contributing factors to AI RMF sub-categories with documented evidence pointers.
ISO/IEC 42001	A.5–A.10 management-system controls	Per-incident control-mapping output suitable for inclusion in the AI management system audit pack.
AU Voluntary AI Safety Standard	Ten Guardrails	Each certificate names the breached guardrail and the deployer accountability path.
United States — sectoral	FDA SaMD, NHTSA ADS, FCA / SEC algorithmic accountability, state-level AI bills (78 across 27 states)	Sector- and state-specific compliance flags with citations to the specific rule, statute, or guidance text.
Singapore — IMDA / MAS FEAT	Model AI Governance Framework, FEAT principles	Principle-by-principle mapping of incident contributing factors.
Canada — AIDA / OSFI E-23	Operational risk for federally regulated financial institutions	OSFI E-23 model-risk control mapping.

Validation, Calibration, and Honest Limits

Metric	Current value	Notes
Primary-party accuracy on the blind benchmark	95.6 %	CALB-1 v0.3, 50 scenarios, three blind rounds
Top-1 match rate	80.0 %	Up from 74 % at v0.4, after M6 (adversarial) and M7 (perception) modules
Mean absolute error on liability apportionment	13.7 percentage points	Down from 15.7 at v0.4, projected to 8 pp after 200 new resolved outcomes
Abstain rate	0 %	Down from 20 %; every scenario now receives a defensible attribution
Damages calibration R-squared	0.86	Calibrated against 14 resolved cases
Wall time (full pipeline)	< 300 ms	Single-thread Node.js 22, no GPU, no LLM call
Wall time per counterfactual perturbation	~ 2 ms	Eighteen perturbations per incident is the default forensic pack
Resolved outcomes corpus	725	209 verified court outcomes, 516 inferred from settlement disclosures
Reliable jurisdiction calibrations	6	United States, United Kingdom, European Union, Australia, Canada, Singapore
Category-specific weight profiles	27	Detected biases in the raw four-factor weights, corrected within ±8 pp
Cross-case correlations	37	Actionable patterns ingested into the calibration blend
Engine size	147,051 LOC TypeScript across 182 modules	Strict mode, Node 22, deterministic, zero ML-runtime dependencies

Honest limits — what the engine does not claim

It does not predict whether a model will fail. The engine is **post-incident**, not pre-incident. Pre-deployment governance products (Credo AI, Holistic AI, Robust Intelligence) and probabilistic eval products (LangSmith, Patronus AI, Galileo, Arize, Weights & Biases) live earlier in the lifecycle and are complementary, not competitive.

It does not replace counsel. The certificate is structured evidence; it is not a legal opinion. Real cases require admissibility analysis under the operating jurisdiction's evidentiary rules (United States: Federal Rules of Evidence Article VII and Daubert; Australia: Uniform Evidence Act sections 76–80 and 144).

The free public demo issues unsigned exemplar certificates (`anchor.status: "demo_ephemeral"`, signature literally `"demo_sig_*"`). Production tenants receive real Ed25519 signatures and a real OpenTimestamps anchor. The free demo is intentionally watermarked so no party can mistake it for a production certificate.

Damages calibration draws on a small (n = 14) verified case set. The R-squared of 0.86 is honest within that sample but the confidence interval is wide on jurisdictions outside the six reliably calibrated.

Threat Model and Adversarial Robustness

The engine assumes an adversary who controls every input field, can replay calls, can attempt to mutate stored ledger entries, and may have temporary access to the operator's signing infrastructure. The defences below are documented in `causallayer-mcp/SECURITY.md`.

Attack	Defence	Detection mechanism
Input fuzzing for liability minimisation	Strict Zod schema; deterministic-only mode rejects under-specified inputs	Returns explicit Zod error; no silent truncation
Replay of an old certificate as new	Every certificate carries `issuedAt`, `prevHash`, and a Bitcoin block reference	Verifier rejects any certificate whose `bitcoinBlockHeight` is greater than the current chain head
Tamper with an old ledger entry	Hash chain plus daily Merkle root plus OpenTimestamps anchor plus public Git mirror	Any of the four layers detects the tamper independently
Exfiltration of the proprietary scoring constants	Constants live server-side; the engine binary is the only artefact ever shipped	Demo tenants receive only the certificate, never the engine internals
Side-channel on the calibration set	Only the calibrated outputs are exposed; the raw resolved-outcome corpus is not part of the certificate	The verifier needs only the public key and the OpenTimestamps proof, not the corpus
Compromised signing key	Daily key-rotation procedure plus a public key-history log in the anchor-log repository	The verifier consults the issuer registry for the relevant `issuedAt` and rejects mismatched signatures
Front-running of `submit_incident` to flood the demo	Per-IP rate cap on the public worker; deterministic-only guardrails	Returns 429 with retry hint; demo certificates are watermarked so flood attacks cannot pollute production
Prompt-injection of the engine	The engine has no LLM in the path	Structurally not applicable
Hallucinated certificates	Any certificate not in the public anchor-log fails verification	`verify_certificate` returns explicit `ANCHOR_NOT_FOUND`

Verification — Thirty Lines of Node.js

A third party can verify any production certificate in roughly thirty lines of Node.js with no FaultKey-side dependency. The procedure:

1. Load the certificate JSON and extract `inputHash`, `outputHash`, `decisionId`, `merkleRoot`, `ed25519Signature`, `ed25519PubKeyFingerprint`, and `openTimestampsProof`.
2. Recompute `decisionId = sha256(inputHash || outputHash || timestamp)` and confirm equality.
3. Fetch the day's Merkle root from the public anchor-log repository (or any mirror).
4. Verify the Merkle inclusion proof for `decisionId` against the published root.
5. Verify the Ed25519 signature on the root against the published key fingerprint for that `issuedAt`.
6. Verify the OpenTimestamps proof against any Bitcoin full node (or any public Bitcoin block-explorer API) and confirm the resulting block hash and height match the certificate's `bitcoinBlockHash` and `bitcoinBlockHeight`.

If all six checks pass, the certificate is mathematically guaranteed to have existed at the recorded `issuedAt`, to have been produced by the recorded engine version on the recorded input hash, to have been signed by the recorded key, and to have been witnessed by Bitcoin. The verifier requires nothing from FaultKey itself.

The reference verifier — `verify-anchor.js` — is shipped MIT-licensed in the public `causallayer-mcp` repository.

Self-Improving Engine Flywheel

The engine is post-incident-first by design but it learns from every resolved outcome. After a case settles, the corresponding certificate is paired with the resolved verdict, and three internal calibration loops update simultaneously:

Calibration target	Mechanism	Effect on next prediction
Cross-case learning	Adjusts the four-factor weights per category from observed primary-party errors	Liability weights shift per category based on what courts actually decide
Platt confidence calibration	Refits the per-jurisdiction sigmoid from the new outcome	Confidence scores become reliable: “70 percent” means correct 70 percent of the time
Damages regression	Updates the R-squared regression coefficients per jurisdiction-and-category	Damage ranges converge toward actual settlement amounts

Before integration, the engine ran on fixed 30/30/20/20 weights, a uniform 70 percent confidence cap, and a 2.7-times damages range. After integration, each of the three loops updates with each new outcome and the engine becomes a closed-loop learning system. Projected metrics after the next 200 resolved outcomes: mean absolute error ~ 8 pp, R-squared ~ 0.94, and reliable jurisdiction calibrations expanded from 6 to 8 or more.

Intellectual Property

The closed-form scoring engine, calibration constants, and outcome corpus are proprietary to FaultKey Protocol. The MCP wrapper, JSON schema, signature verifier, certificate format, and reference engine impl are open-source under MIT.

Filed protections:

- **AU Standard Patent 2026202546** — filed 2 April 2026.
- **AU Provisional Patent 2026903079** — filed 3 April 2026.
- Fourteen further patent applications in preparation covering causal-graph construction for AI liability, multi-agent fault apportionment, self-calibrating confidence scoring, jurisdiction-adaptive liability weighting, regulatory-compliance mapping, and the counterfactual simulator architecture (running deterministic attribution modules in a perturbation loop to prove legal but-for causation).

Endpoints, Repositories, and Canonical URLs

Resource	URL
Site	https://faultkey.com
MCP demo	https://mcp.faultkey.com/mcp
MCP healthcheck	https://mcp.faultkey.com/healthz
REST mirror	https://api.faultkey.com/v1/incidents/analyze
Try-in-browser	https://faultkey.com/try
Verify-a-cert	https://faultkey.com/verify
Worked case study	https://faultkey.com/case-study/refund-bot
Compared with competitors	https://faultkey.com/vs
Pricing	https://faultkey.com/pricing
Press kit	https://faultkey.com/press
Transparency disclosure	https://faultkey.com/transparency
Changelog	https://faultkey.com/changelog
MCP server repo (MIT)	https://github.com/smq9sn5jck-coder/causallayer-mcp
Anchor-log repo (public)	https://github.com/smq9sn5jck-coder/causallayer-anchor-log
Pre-genesis notice	https://github.com/smq9sn5jck-coder/causallayer-anchor-log/blob/main/PRE-GENESIS-NOTICE-2026-05-15.md
Contact	hello@faultkey.com

Provenance of This Document

This brief is generated from the canonical sources inside the repositories listed above and reflects the engine state at v0.5.0, anchor schema v1, and ledger schema v1 on 17 May 2026. The brief itself is committed to the public `faultkey-landing` repository at `docs-src/faultkey-engine-brief.md` and rendered to PDF at `client/public/docs/faultkey-engine-brief.pdf`. Every revision is logged in `/changelog`, mirrored in the anchor log repo, and re-archived to the Software Heritage and Internet Archive.

For the most recent revision, fetch directly:

> *https://faultkey.com/docs/faultkey-engine-brief.pdf*

For deep technical questions, please contact `hello@faultkey.com`.