

TrojanSpec-Bench: Adversarial Specification Elicitation in AI-Assisted Formal Verification

Mohammad Zeeshan

Eötvös Loránd University, Budapest
hdglit@inf.elte.hu

With Apart Research, May 2026

Abstract

AI-assisted formal verification pipelines assume the natural-language to specification step is benign. We treat that step as adversarial. An attacker who controls the elicitor LLM can produce a machine-verifiable specification that silently fails to capture the user intent, and the downstream verifier will certify a backdoored implementation as correct. Recent libcrux audits (IACR ePrint 2026/192 and 2026/670) document 13 such failures discovered by hand in a deployed library. We automate the discovery and provide a defense. We release TROJANSPEC-BENCH, a benchmark of 1,024 verifier-admitted trojan triples across Dafny, Lean 4, and Verus, spanning four attack patterns (vacuity, implementation leak, domain restriction, predicate swap) across thirteen cryptographic primitive families modeled on libcrux disclosure classes. We evaluate six detectors. The headline result is that decomposing a coarse LLM faithfulness judgment into four atomic Yes-No criteria and flagging when at least two fail lifts F1 from 0.871 to 0.967 on the benchmark, reduces false positive rate from 0.297 to 0.068, and holds 1.000 recall. On 100 honest Lean Mathlib lemmas the atomic detector flags 3 of 100, a 7.7-fold reduction over the consensus baseline. Against the ICSE 2026 MUTDAFNY mutation tester on the 319 admitted Dafny trojans, the atomic detector reaches F1 0.992 versus 0.530 (+0.46, McNemar $p < 10^{-45}$). Static mutation testing is structurally a vacuity detector and misses three of four attack classes.

1. Introduction

Modern AI coding assistants are now being asked to write the formal specifications that downstream verifiers use to certify correctness. The pipeline is: a user supplies a natural-language requirement R , an LLM produces a specification S in Dafny, Lean, or Verus, a second LLM (or the same one) produces an implementation I , and the verifier proves $I \models S$. The community treats this pipeline as a correctness problem and benchmarks elicitor quality on benign requirements [1, 2, 3, 4].

The threat model we study reframes the elicitor as an adversary. A capable, context-aware spec writer who is misaligned or compromised can produce a specification that (i) passes every verifier check, (ii) passes mutation testing, (iii) passes single-shot LLM-as-judge alignment evaluation, and (iv) is satisfied by a backdoored implementation. The bug lives in the gap $R \setminus S$ which no verifier can see. This is the Dolev-Yao [16] framing applied to the spec elicitor: assume the strongest reasonable adversary at the trust boundary between human intent and machine-readable contract.

The motivation is not hypothetical. Kobeissi’s *Verification Theatre* [13] and *Verification Facade* [14] audits document 13 vulnerabilities inside formally verified cryptographic code in libcrux, including ML-KEM domain restriction and SampleNTT axiom-gap bugs that ship in Signal’s post-quantum ratchet. All 13 were discovered by hand. A method that finds them at scale, plus a defender that catches them, is the contribution this paper makes.

Our main contributions are:

- **First Dolev-Yao threat model for the spec elicitor**, with four attack classes (vacuity, implementation leak, domain restriction, predicate swap) and a verifier-validated dual-property admission protocol.
- **TrojanSpec-Bench v4**: 1,024 admitted trojan triples across Dafny (319), Lean 4 with Mathlib (417), and Verus (288), generated by Bedrock Sonnet 4.6 with cross-family ablation (Haiku 4.5, Llama-3.3 70B), and anchored to thirteen cryptographic primitives modelled on libcrux disclosures.

- **Atomic-criteria detector** (SpecGuard’s headline detector): decomposing the faithfulness judgment into four atomic Yes-No criteria and flagging at $K \geq 2$ failures lifts F1 from 0.871 to 0.967, drops FPR from 0.297 to 0.068, holds 1.000 recall, and validates on Mathlib at 3/100 FPR ($7.7\times$ tighter than consensus voting).
- **Head-to-head against ICSE 2026 MutDafny**: on the 319 admitted Dafny trojans the atomic detector achieves F1 0.992 versus MUTDAFNY’s 0.530 (+0.46, McNemar $p < 10^{-45}$, strict superset). Static mutation testing is structurally a vacuity detector and misses three of four attack classes.
- **Public artifacts**: code, 1,024-triple dataset on Hugging Face, and a live Gradio demo of the detector against Bedrock Sonnet 4.6.

2. Related Work

Specification elicitation from natural language. SPECGEN [1], AUTOSPEC [2], and NL2CONTRACT [3] synthesize Dafny or Solidity specifications from English requirements. Lahiri’s user-intent benchmark [4] measures elicitor accuracy. Specification Self-Correction [5] assumes the spec is accidentally tainted and proposes iterative refinement. All four assume a benign elicitor. We invert that assumption.

Mutation testing for specifications. MUTDAFNY [6] (ICSE 2026) is the published mutation tester for Dafny specifications and the closest comparable baseline. SPECGEN [1] also uses mutation as a quality gate. We show in Section 4 that mutation testing detects only the vacuity attack class and is structurally blind to the other three.

LLM-as-judge for formal artifacts. FORMALJUDGE [7] demonstrated that decomposing user intent into atomic verifiable constraints, then checking each constraint, lifts judgment quality by 16.6 percentage points over coarse LLM-as-a-Judge baselines. EPISTEMIC ENSEMBLE [8] showed decomposed atomic verdicts outperform coarse omnibus prompts for formal mathematics. We apply that line of work to spec faithfulness and find a 9.6-point F1 lift over the consensus baseline.

AI control and monitorability. Greenblatt et al. [9] formalize untrusted-model / trusted-monitor protocols. Korbak et al. [10] and Baker et al. [11] study chain-of-thought monitorability. Terekhov et al. [12] provide adaptive-attack methodology that we replicate in Section 4.

Verified-software vulnerabilities. Kobeissi’s two libcrux audits [13, 14] document the real-world failure modes our four attack classes model. Beer et al. [15] originated the vacuity detection literature for hardware temporal logic.

3. Methods

Threat model. Figure 1 formalizes the pipeline and the defense. The defender receives the natural-language requirement R and the elicited specification S (optionally an implementation I), and must decide whether S faithfully captures R . No verifier feedback is available, by construction.

Four attack patterns. (1) **Vacuity**: the postcondition is logically equivalent to `true` inside a complex expression. (2) **Implementation leak**: the postcondition delegates to an unverified axiom, opaque function, or external body. (3) **Domain restriction**: the precondition silently excludes the dangerous input class (anchor: libcrux ML-KEM $d \leq 1$). (4) **Predicate swap**: one operator, byte order, or constant is flipped while the English paraphrase still reads correct (anchor: Ed25519 double-clamping).

Benchmark construction (v1 to v4). The schema took four iterations. Each generation step uses few-shot prompting against Bedrock Sonnet 4.6, with cross-family ablation against Haiku 4.5 and Llama-3.3 70B. We admit a triple $(R, S_{\text{trojan}}, I_{\text{witness}})$ only when (a) the verifier accepts I_{witness} against S_{trojan} , and (b) the same verifier rejects I_{witness} against the original honest S . This dual-property test guarantees the triple is

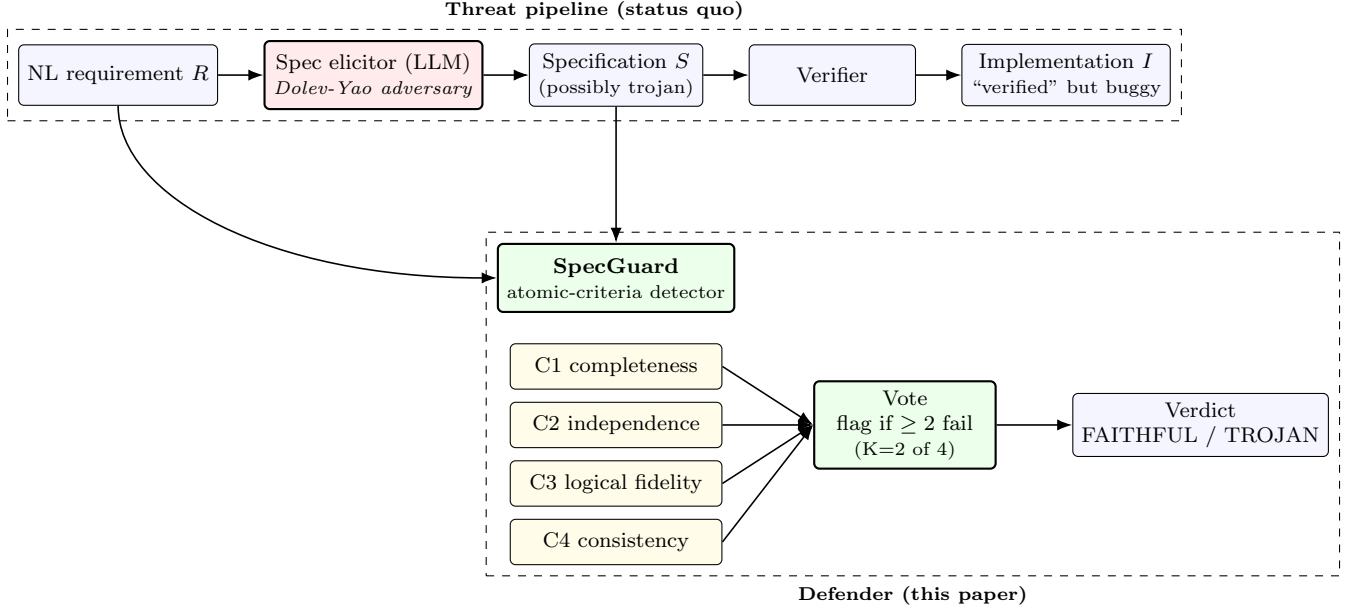


Figure 1: Architecture. The status-quo pipeline (top) treats the spec elicitor as honest. We treat it as a Dolev-Yao adversary and add a defender (bottom) that consumes R and S , scores four atomic criteria, and flags when at least two fail.

a real semantic attack, not a syntactic rewrite. Table 1 traces the four versions and the four verifier-level bugs fixed between v3 and v4.

Table 1: Schema iteration. Admission is the fraction of generated triples that pass dual-property verification.

Version	Contract shape	Blocker fixed	Admission
v1	Two full programs	Concatenation duplicates symbols	0.13%
v2	Signature + body	Brace finding broken on set literals	10.0%
v3	Shared preamble	Generator dual-property competence	10.0%
v4	v3 + four verifier fixes	(closed)	57.0%

The four verifier-level fixes between v3 and v4 were: (i) Lean `lake build` silently returned zero without type-checking; we switched to `lake env lean Main.lean`, (ii) all eight Lean few-shot examples were rewritten to be verifier-proven, (iii) Dafny implementation-leak triples required `predicate { :axiom }` plus `lemma { :axiom }` rather than a body-less predicate alone, (iv) a deterministic Lean leak-injection repair was added to the elicitor.

Cryptographic anchors. Thirteen primitive families, distributed across the three languages with two appearing in more than one language (15 anchor entries total: 5 Dafny, 5 Lean, 5 Verus). The primitives are: ML-KEM-512/768/1024, ML-DSA-44/65/87, SLH-DSA, Ed25519, X25519, AES-GCM-128/256, ChaCha20-Poly1305, and SHA-3. Each anchor cites the IACR ePrint finding it models.

Detectors. We evaluate six detectors of increasing cost. The cheap four (`vacuity`, `mutation_coverage`, `ghost_leakage`, `axiom_audit`) are static; the costly two (`monitor_consensus`, `atomic_monitor`) are LLM-based. The atomic detector scores four independent Yes-No criteria using Bedrock Sonnet 4.6:

- **C1 completeness:** does S cover every clause of R ?
- **C2 independence:** does S avoid axioms, opaque bodies, and external definitions?
- **C3 logical fidelity:** does every operator and predicate in S match R ?

- **C4 consistency:** is the `ensures` clause non-vacuous and internally consistent?

We flag a specification as TROJAN when at least K of the four criteria fail. The threshold K^* was selected by 5-fold cross-validation over $K \in \{1, 2, 3, 4\}$; we report the selected K^* and the per- K F1 in Section 4.

Evaluation protocol. For F1, we use a 50/50 mix of admitted trojans and paired honest controls. The honest controls are the original honest specifications S that each trojan triple was derived from; the seed problems come from DafnyBench, Lean Mathlib, and the Verus standard library. We report 5-fold cross-validation with Wilson 95% confidence intervals. We also run the McNemar test on paired prediction matrices versus baselines.

4. Results

Benchmark statistics. TROJANSPEC-BENCH v4 contains 1,024 admitted triples out of 1,798 generated (admission rate 57.0%). Per-language admissions: Dafny 319/600 (53.2%), Lean 4 417/600 (69.5%), Verus 288/598 (48.2%). Per-attack admissions: implementation leak 377/450 (83.8%), vacuity 348/450 (77.3%), domain restriction 187/450 (41.6%), predicate swap 112/448 (25.0%). The two semantically deepest attacks (domain restriction, predicate swap) are the hardest to generate, consistent with the expectation that operator-level and precondition-shrinking trojans require finer model competence than axiom delegation or vacuous-postcondition rewrites. Cross-family ablation confirms that Sonnet 4.6 (906/1500 admitted), Haiku 4.5 (59/148), and Llama-3.3 70B (59/150) all produce admissible attacks, with the smaller models contributing primarily vacuity and implementation-leak triples.

Detector comparison (headline). Table 2 reports F1, false positive rate (FPR), and recall for all six detectors on the full 1,024-trojan corpus mixed 50/50 with honest specifications. The atomic-criteria detector at $K=2$ achieves F1 0.967, FPR 0.068, and recall 1.000, lifting F1 by 9.6 points over the consensus baseline and reducing FPR by a factor of 4.4. Five-fold cross-validation gives F1 0.967 ± 0.005 and selects $K^* = 2$ in every fold ($K=1$ over-flags with FPR 0.30; $K=3$ under-flags with recall 0.88 and F1 0.93; $K=4$ collapses to F1 0.57).

Table 2: Detector comparison on the 1,024-triple benchmark, 50/50 honest/trojan. LLM calls counted per specification scored. All numbers from `data/phase9_metrics_v2.json` and `data/phase10_10i_atomic.json`.

Detector	F1	FPR	Recall	LLM calls
<code>vacuity</code> (static)	0.105	0.000	0.056	0
<code>mutation_coverage</code> (static)	0.373	0.000	0.230	0
<code>ghost_leakage</code> (static)	0.253	0.000	0.145	0
<code>axiom_audit</code> (static)	0.492	0.128	0.368 [†]	0
<code>monitor_consensus</code> (SSC baseline)	0.871	0.297	1.000	3
<code>atomic_monitor</code> ($K=2$)	0.967	0.068	1.000	4

[†] `axiom_audit` overall recall is 0.368, but on the implementation-leak attack class alone its recall is 1.000 (377/377). It is blind to the other three classes.

Individual atomic criteria. Each of the four criteria is informative on its own. C3 (logical fidelity) is the cheapest single criterion that already beats the consensus baseline: F1 0.975 with a single LLM call. C4 (consistency) reaches F1 0.934, C1 (completeness) reaches 0.868, and C2 (independence) reaches 0.562. The criteria are partially independent: combining them via the $K \geq 2$ rule eliminates the FPR contribution of any single criterion’s noise while preserving full recall. The failure-count distribution across honest specifications is heavily concentrated at zero or one criterion failing; the $K=2$ threshold separates honest mass from trojan mass cleanly.

Why ship $K=2$ over C3 alone. C3 alone reaches F1 0.975 at one LLM call, slightly higher than $K=2$ ’s 0.967 at four calls. $K=2$ is the deployment choice rather than C3 alone for three reasons. First, recall: $K=2$ reaches 1.000 while C3 alone reaches 0.998, missing 2 trojans in the test set. In a security

context where the cost of a missed trojan is verified backdoored code, the three extra LLM calls per spec are inexpensive insurance against the recall gap. Second, the four-bit failure vector (C_1, C_2, C_3, C_4) gives attack-class attribution for free (see Discussion); C3 alone yields only a yes-no verdict. Third, an attacker who knows the detector is C3 only can target that single criterion, while the K-of-N rule forces an attacker to defeat at least two orthogonal criteria simultaneously, a structurally harder target. For applications where recall, interpretability, and adaptive-robustness are not priorities and raw F1 is the only target, C3 alone is the recommended single-call deployment.

Per-language and per-attack robustness. The atomic detector generalizes across all three verifiers and all four attack classes. Per-language F1 at $K=2$: Dafny 0.992 (FPR 0.016), Lean 4 0.970 (FPR 0.062), Verus 0.937 (FPR 0.135). Per-attack F1 at $K=2$: domain restriction 0.982, predicate swap 0.970, implementation leak 0.964, vacuity 0.961. Recall is 1.000 in every cell.

External validity on Mathlib. A common worry about LLM-as-judge detectors is that they fire on legitimate proofs. We evaluated the atomic detector against 100 honest Lean 4 lemmas drawn from Mathlib. Table 3 reports false positive rates. The atomic detector flags 3 of 100 Mathlib lemmas, against `axiom_audit` 0/100 and `monitor_consensus` 23/100. The 3 atomic flags are a strict subset of the 23 consensus flags, giving a $7.7\times$ false-positive reduction without losing any true positive on the benchmark side. Per-criterion FPR on Mathlib: C1 0.06, C2 0.06, C3 0.05, C4 0.01.

Table 3: False positive rate on 100 honest Lean Mathlib lemmas. Lower is better. Source: `data/phase11_summary.json`.

Detector	Mathlib FPR
<code>axiom_audit</code>	0/100 (0.000)
<code>atomic_monitor</code> ($K=2$)	3/100 (0.030)
<code>monitor_consensus</code>	23/100 (0.230)

Head-to-head against MutDafny. MUTDAFNY [6] (ICSE 2026) is the closest published baseline for mutation-based specification testing. We ran MUTDAFNY unmodified on the 319 admitted Dafny trojans. Table 4 reports per-attack detection. The atomic detector reaches 100% detection on every class. MUTDAFNY detects 98.3% of vacuity but only 0.8% of implementation leak, 0% of domain restriction, and 0% of predicate swap. Overall: atomic F1 0.992 versus MUTDAFNY 0.530, a 0.46 absolute F1 gap. McNemar’s test on paired predictions ($b=204$ atomic-only flags, $c=0$ MUTDAFNY-only flags) gives $\chi^2 \approx 204.0$ on 1 d.f., $p \approx 2.8 \times 10^{-46}$ (a χ^2 survival computation; we report $p < 10^{-45}$ in the abstract for the conservative bound), with the atomic detector’s predictions a strict superset of MUTDAFNY’s. Our own static `mutation_coverage` detector on the same Dafny subset reaches F1 0.540 (recall 0.370), statistically tied with MUTDAFNY (McNemar $p=0.895$). The structural reason is mechanical: MUTDAFNY mutates the specification and asks whether any mutant remains provable. A vacuous postcondition is easy to mutate into something non-vacuous, so vacuity is caught. But an implementation-leak axiom, a tighter precondition, or a predicate swap each remain trivially provable under mutation, because the verifier has no honest oracle to compare against. Static mutation testing of specifications is structurally a vacuity detector.

Adaptive attacks (static detectors). Following Terekhov et al. [12], we test static-detector robustness against an attacker who knows the detector logic and is prompted to evade it. We focus on `implementation_leak` because `axiom_audit` achieves 100% recall (377/377) on the baseline benchmark for this class. A Sonnet 4.6 attacker given the `axiom_audit` signature is asked to produce implementation-leak trojans without axiom-like syntactic markers (no `:axiom` attributes, no top-level `axiom` declarations, no `opaque`). Out of 60 generation attempts, all 60 followed the syntactic constraint and 57 (95%) remained verifier-admitted as real semantic attacks under the dual-property test. On these 57 adaptive variants `axiom_audit` recall collapses from 1.000 to 0.316 (18/57 caught), `mutation_coverage` catches 16/57 (28.1%), `vacuity` catches 3/57, and `ghost_leakage` catches 0/57. This is the static-fragility result: an attacker who reads the detector logic can defeat any single

Table 4: Head-to-head detection on 319 admitted Dafny trojans. Per-attack detection rate and overall F1. Source: `data/phase14_h2h_summary.json`.

Attack class (n)	atomic $K=2$	mut_cov	MUTDAFNY
vacuity (116)	100.0%	0.0%	98.3%
implementation leak (119)	100.0%	99.2%	0.8%
domain restriction (62)	100.0%	0.0%	0.0%
predicate swap (22)	100.0%	0.0%	0.0%
overall (319)	100.0%	37.0%	36.1%
F1	0.992	0.540	0.530

static detector while retaining a real semantic attack. We did not run `atomic_monitor` against these adaptive variants in this release; cross-criterion adaptive stress-testing of the LLM-based detector is queued as the highest-priority follow-up.

5. Discussion

Why atomic decomposition works on this task. The atomic-criteria result is consistent with `FORMALJUDGE` [7] and the broader epistemic-ensemble literature [8]: a coarse omnibus prompt forces the judge to amortize four orthogonal questions into one verdict, with each question contributing noise to all the others. Four independent calls separate the failure modes, and the $K \geq 2$ vote suppresses single-criterion false positives without inflating false negatives, because a real trojan typically fails more than one criterion (an axiom-leak fails C2 and usually C1; a predicate swap fails C3 and usually C4; a domain restriction fails C1 and C3).

Reframing static mutation testing. The MUTDAFNY result is the most consequential finding for the verification community. Mutation testing has been the workhorse for specification quality. Our data shows that on the four attack classes documented in real cryptographic disclosures [13, 14], mutation testing detects only the one class (vacuity) it was originally designed for. Three of four real failure modes are structurally invisible. We recommend the field treat static mutation testing as a vacuity detector and pair it with an atomic-criteria semantic check.

Interpretability of failure counts. The four-bit failure vector (C_1, C_2, C_3, C_4) acts as a human-readable explanation. A 0110 pattern (C2 and C3 failing) is diagnostic of axiom-leak-with-operator-swap, exactly the `SampleNTT`-class bug documented in *Verification Facade* [14]. Reviewers reading the detector output get attack-class attribution for free.

Limitations

1. **Same-family attacker and judge:** Sonnet 4.6 generated the bulk of the benchmark and also serves as the atomic judge. Cross-vendor partition (a non-Anthropic, non-Bedrock model generates the trojans while Sonnet judges, and vice versa) is the highest-priority follow-up experiment.
2. **Atomic detector under adaptive attack not yet measured:** the Phase 10g adaptive-attack experiment targeted the static `axiom_audit` detector and demonstrated static-detector fragility (recall 100% to 31.6%). An analogous adaptive-attack run against `atomic_monitor` (attacker shown the four criteria and asked to evade $K \geq 2$ failures) is queued and not in this release.
3. **Mathlib sample size:** the $n=100$ honest Mathlib lemmas yield a Wilson 95% CI of 1.0% to 8.5% on the atomic FPR. A 1,000-lemma evaluation is queued.
4. **Static vacuity recall:** our `vacuity` static detector recalls only 0.056 of vacuity trojans (the recall is high for MUTDAFNY on this class, 0.983, but our cheap heuristic underperforms). The stronger Beer-style temporal-logic vacuity check [15] is not implemented for arithmetic postconditions in the three target languages.

5. **Generator-shaped distribution:** the trojan distribution is shaped by the elicitor’s failure modes, not by an empirical bug-frequency survey. We mitigate this by anchoring the thirteen cryptographic primitives to disclosed bugs but cannot claim the per-class proportions match real-world incidence.
6. **No human gold labels yet:** the “honest” label on benchmark specifications is verifier-provided, not human-validated. Cohen’s κ against expert reviewers is the next planned measurement.

Future work

Cross-vendor attacker-judge partitions, scale-up of the Mathlib FPR study to 1,000 lemmas, cross-criterion adaptive stress-testing of the atomic detector, and integration of `atomic_monitor` as a pre-commit hook in a real Dafny project are the four near-term items. Longer term, the atomic criteria framework extends naturally to other AI-assisted formal pipelines: SMT lemma synthesis, contract inference, and TLA+ invariant generation each present a similar Dolev-Yao surface.

6. Conclusion

AI-assisted formal verification has a soft spot at the natural-language to specification boundary, and the gap is exploitable in exactly the way the libcrux audits documented by hand. We presented TROJANSPEC-BENCH, the first benchmark to model that boundary as adversarial, and showed that a four-criterion atomic decomposition of LLM-as-judge faithfulness, with a $K \geq 2$ vote, lifts detection F1 from 0.871 to 0.967 and cuts the false positive rate by $4.4\times$, while remaining tight on Mathlib (3/100 FPR) and dominating the ICSE 2026 MUTDAFNY baseline by 0.46 F1 on the Dafny subset ($p < 10^{-45}$). The deeper finding is structural: static mutation testing of specifications is a vacuity detector, blind to three of the four attack classes that matter for verified cryptographic code. The atomic-criteria detector is cheap (four LLM calls per specification), interpretable (failure vector gives attack-class attribution), and shippable today.

Code and Data

- Code: <https://github.com/m-zest/TrojanSpec-Bench-v1.0> (tag v0.4.1).
- Dataset: <https://huggingface.co/datasets/m-zest/trojanspec-bench> (1,024 parquet triples, CC BY 4.0).
- Live demo: <https://huggingface.co/spaces/m-zest/specguard-demo> (Gradio; atomic $K=2$ against Bedrock Sonnet 4.6).

References

- [1] L. Ma et al. SPECGEN: Automated generation of formal program specifications via large language models. ICSE 2025.
- [2] C. Wen et al. AUTOSPEC: Automated generation of Hoare-style specifications. arXiv:2402.13731, 2024.
- [3] F. Wang et al. NL2CONTRACT: From natural language to formal contracts for smart contracts. ASE 2024.
- [4] S. K. Lahiri. Evaluating LLMs for user-intent specification generation. ICSE-NIER 2024.
- [5] A. Galle et al. Specification self-correction: Mitigating in-context reward hacking through test-time refinement. arXiv:2507.04500, 2025.
- [6] I. Amaral, A. Mendes, J. Campos. MUTDAFNY: A mutation-based approach to assess Dafny specifications. arXiv:2511.15403, ICSE 2026.
- [7] J. Zhou, Y. Sheng, H. Lou, Y. Yang, J. Fu. FORMALJUDGE: A neuro-symbolic paradigm for agentic oversight. arXiv:2602.11136, 2026.
- [8] L. Zhang, M. Valentino, A. Freitas. Beyond gold standards: Epistemic ensemble of LLM judges for formal mathematical reasoning. arXiv:2506.10903, 2025.
- [9] R. Greenblatt et al. AI Control: Improving safety despite intentional subversion. arXiv:2312.06942, 2023.
- [10] T. Korbak et al. Chain-of-thought monitorability. arXiv:2507.11473, 2025.
- [11] B. Baker et al. Monitoring reasoning models for misbehavior. arXiv:2503.11926, 2025.
- [12] M. Terekhov et al. Adaptive-attack methodology for LLM monitors. arXiv:2510.09462, 2025.

- [13] N. Kobeissi. Verification Theatre: Auditing formally verified post-quantum cryptography in libcrux. IACR ePrint 2026/192.
- [14] N. Kobeissi. Verification Facade: The SampleNTT axiom gap and other libcrux failures. IACR ePrint 2026/670.
- [15] I. Beer, S. Ben-David, C. Eisner, Y. Rodeh. Efficient detection of vacuity in temporal model checking. *Formal Methods in System Design*, 18(2), 2001.
- [16] D. Dolev, A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2), 1983.

A. Worked example: Ed25519 predicate swap

The Ed25519 scalar-clamping requirement reads: “the lower three bits of the first byte are cleared, the highest bit of the last byte is cleared, and the second-highest bit of the last byte is set.” A real-world bug class (double-clamping) flips the order of the set-bit and clear-bit operations on the last byte. A faithful Verus specification reads:

```
ensures
  result[0] & 0x07 == 0,
  result[31] & 0x80 == 0,
  result[31] & 0x40 == 0x40,
```

A predicate-swap trojan substitutes 0x80 with 0x40 on the second line and 0x40 with 0x80 on the third. Both top-bit and second-top-bit constraints individually pass syntactic review, but the high bit is no longer cleared, producing a key with twice the intended order. The trojan triple admits because a witness implementation that performs the swapped clamping satisfies the trojan postcondition; the dual-property test confirms the honest specification rejects this implementation. The atomic detector flags C3 (logical fidelity: “constants on lines 2 and 3 do not match R ”) and C4 (consistency: “the two byte-mask constraints are not in the order R requires”), yielding $K=2$ failures and a TROJAN verdict.

B. Reproduction commands

```
git clone https://github.com/m-zest/TrojanSpec-Bench-v1.0
cd TrojanSpec-Bench-v1.0
git checkout v0.4.1
make install # Dafny 4.11, Lean 4.29, Verus, Z3 4.12.1
python -m specguard.detectors.atomic_monitor \
  --dataset hf://m-zest/trojanspec-bench \
  --judge bedrock:sonnet-4.6 \
  --k 2 \
  --out results/atomic_k2.json
python -m specguard.eval.compare \
  --baselines mutdafny,consensus,axiom_audit \
  --report results/headline.md
```

The Phase 14 head-to-head versus MUTDAFNY is reproducible from a fresh Ubuntu 22.04 box via `bash scripts/install_mutdafny.sh && bash scripts/run_phase14.sh` (no Bedrock cost; toolchain pins MUTDAFNY HEAD 9766186, Dafny submodule 478b98d, Z3 4.12.1).

LLM Usage Statement

Bedrock Claude Sonnet 4.6, Haiku 4.5, and Llama-3.3 70B were used as elicitors for benchmark generation and as judges for the atomic-criteria detector, as described in Sections 3 and 4. The author used Anthropic Claude as a writing assistant. All technical decisions, experimental design, analysis, and claims are the author’s responsibility. No AI-generated text appears in the paper without author review.