Axiom-Audited Trustworthy Formalization of Game-Theoretic Commitment in Lean

Jan Ondras [1]

Abstract

As large language models increasingly *generate* proofs in Lean, Rocq, and Isabelle, a machine-checked proof's trustworthiness turns on two questions its script does not foreground: which foundational axioms it rests on, and where an external, unproved trust assumption enters. We answer both mechanically in a Lean 4 formalization of a game-theoretic commitment result, and from it extract two methods for *trustworthy formalization* that we argue transfer to verifying AI-generated and autoformalized mathematics. The first treats *axiom hygiene as an executable regression test*: `#guard_msgs`-guarded `#print axioms` assertions pin the foundational-axiom footprint of all 56 declarations (kernel+Classical only; no `sorry`, no custom axioms), so any drift fails the build with a mismatch. The second isolates the development's single trust assumption (that an external attestation faithfully reflects intent) in a `TrustedOracle` typeclass with an explicit truthfulness predicate, a worked instance, and a non-instance counterexample proving the obligation is non-vacuous. The case-study object is a dominance-and-equilibrium theorem: for every Stag-Hunt-ordered game, routing players through a verified `wrap` combinator makes attesting Cooperate weakly dominant, with mutual cooperation the unique Pareto-undominated Nash equilibrium. We prove this for the 2-player and n-player, symmetric and asymmetric cases, the 2-player results obtained by machine-checked specialization of the n-player ones.

[1]Massachusetts Institute of Technology, Cambridge, MA, USA. Correspondence to: Jan Ondras <janko@mit.edu>.

The 3rd *AI for Math Workshop* at the 43rd International Conference on Machine Learning (ICML), Seoul, South Korea, 2026.
Copyright 2026 by the author(s).

1. Introduction

A machine-checked proof is only as trustworthy as the assumptions beneath it, and two of those assumptions are the ones a proof script does not display: which foundational axioms it ultimately invokes, and whether any unproved hypothesis is folded into its definitions. As large language models increasingly produce formal proofs, auditing both has become a central reliability concern for formalized and autoformalized mathematics alike. A Lean development can type-check yet silently depend on a strong axiom, admit a `sorry`, or bake an empirical hypothesis into a definition: failures that are both harder to spot and higher-stakes when the proof was machine-generated. This risk is not unique to machine-generated proofs: in any actively developed library a routine refactor, or a newly imported lemma that itself rests on a strong axiom or `native_decide`, can enlarge a declaration's axiom footprint while its statement is unchanged, so the question of *what a proof depends on* is independent of whether a human or a model wrote it, though machine generation makes it more pressing.

We study these two reliability questions through a concrete formalization target drawn from game theory. Program-equilibrium theory (Tennenholtz, 2004; Kalai et al., 2010; Barász et al., 2014; Critch, 2019; Oesterheld, 2019) shows that binding *commitment devices* can restore cooperation in coordination games: if each player's action is made to depend on a verified property of the others', the dominant action becomes Cooperate. The theory is mature but pen-and-paper (Tennenholtz, 2004; Kalai et al., 2010; Sauerberg & Oesterheld, 2026) or predicated on adversarial source-code mutual reading (Barász et al., 2014; Critch, 2019); none of it is machine-checked. We choose a commitment-device theorem precisely because it carries a non-trivial *trust assumption* to separate (a purely arithmetic lemma would exercise neither the trust-boundary nor the axiom-footprint method), and because its dominance and equilibrium statements are clean enough to formalize completely and audit in full.

The object we formalize is a thin *handshake* wrapper that mediates between $n \geq 2$ players through an external attestation surface: each player posts a binding attestation ("I will play C") before acting, and the wrapper realizes

action C iff every attestation agrees on C, otherwise D. The resulting meta-game makes attesting C weakly dominant for every player, with mutual cooperation the unique Pareto-undominated Nash equilibrium.[1] We develop the two-player case first (§3) and then the arbitrary-n case in §4.

A second design principle runs through the formalization: we treat the strategic combinator and the oracle-trust assumption as *separately reviewable* artifacts. The dominance proof is parameterized over arbitrary Stag-Hunt-ordered payoffs and oracle-agnostic by construction; the oracle threat model is discharged at the deployment surface (smart contracts, trusted-execution-environment (TEE) attestation, decentralized identifiers / verifiable credentials (DIDs/VCs)), not inside the proof.

Our contributions are a fully audited case study (C1) and the two transferable methods it demonstrates (C2, C3): **(C1) A machine-audited formalization.** We mechanize in Lean 4 a commitment-induced dominance-and-equilibrium theorem over arbitrary Stag-Hunt-ordered games (2-player and n-player, symmetric and asymmetric), `sorry`-free and with no custom axioms, the 2-player results obtained as verified specializations of the n-player development. To our knowledge this is the first Lean 4 mechanization of a commitment-device strategic-incentive transformation with a machine-audited axiom footprint and an explicit trust-boundary typeclass (§2); the complete machine-checked development is openly available (Ondras, 2026). **(C2) Axiom hygiene as an executable regression test.** Build-time `#guard_msgs`-guarded `#print axioms` assertions pin the foundational-axiom footprint of all 56 declarations, turning "what does this proof depend on?" into a check that fails the build on any drift. This is a lightweight, transferable practice for any development whose trustworthiness matters, and, we argue, for pipelines that verify AI-generated proofs (§5, D3). **(C3) An explicit trust-boundary design pattern.** The development's single trust assumption is isolated in a `TrustedOracle` typeclass with an explicit `truthful` predicate, kept disjoint from the dominance proof, with a worked instance and a non-instance counterexample witnessing that the obligation is non-vacuous, so a reader audits where trust enters, separately from the proved core.

[1]The all-defector attestation profile is also a (weak) Nash equilibrium of the wrapped meta-game (unilaterally switching to C leaves the wrapper output at D); more generally, for $n \geq 3$ every profile with at least two defectors is a weak Nash equilibrium, since no single deviation can flip the wrapper to C while another defector remains. All such profiles are Pareto-dominated by all-C, and the ε-incentive of Corollary 4.3 removes them as equilibria, leaving all-C the unique strict equilibrium. That all-C is a Nash equilibrium is itself mechanized (`wrapN_weak_nash`); the weak-equilibrium remarks for the non-cooperative profiles follow from the wrapper's binary collapse and are stated rather than separately mechanized.

2. Background and Related Work

Mechanized game theory and proof engineering. Our development is built in Lean 4 (de Moura & Ullrich, 2021) and uses *mathlib* (The mathlib Community, 2020) only for rational arithmetic, `linarith`, and finite-type support for the n-player development. Prior proof-assistant game theory targets equilibrium existence or fixed mechanisms: Bagnall et al. (2017) give a Coq library for algorithmic game theory (Nash, smooth games, price-of-anarchy bounds); Le Roux et al. (2017) prove Nash existence in Coq and Isabelle; Pomeret-Coquot et al. (2023) formalize incomplete-information games in Coq; Caminati et al. (2015) mechanize Vickrey auctions in Isabelle; and on the Lean 4 side MixedMatched contributors (2024) formalize the Prisoner's Dilemma and Nash Demand. Unlike these, we mechanize a *strategic-incentive transformation*: a commitment device that changes which action is dominant. We additionally make the development's foundational-axiom footprint and its single trust boundary machine-auditable, an emphasis absent from all of them. To our knowledge no prior proof-assistant work mechanizes a commitment-device dominance theorem.

Program equilibrium and commitment folk theorems. Tennenholtz (2004) showed that if agents can read each other's source code, program-equilibrium payoffs include mutual cooperation in the Prisoner's Dilemma; Kalai et al. (2010) generalized this with a commitment folk theorem. Barász et al. (2014) obtained robust cooperation via provability logic (FairBot, PrudentBot), but in modal logic on paper with no proof-assistant mechanization; Critch (2019) and Oesterheld (2019) add bounded-resource and robust variants. The closest pen-and-paper predecessor is Sauerberg & Oesterheld (2026) (*ex post verifiable commitments* for safe Pareto improvements). We differ in being *ex ante*, built for *opaque populations* where source-code reading is infeasible, and machine-checked with build-time axiom-hygiene assertions rather than pen-and-paper. Our `wrap` combinator also differs from a cryptographic commitment scheme (which binds a *hidden* value for later opening) and from Byzantine-fault-tolerant consensus (which secures agreement among faulty replicas): we assume a single trusted attestation oracle, isolate that assumption in `TrustedOracle`, and defer a fault-tolerant quorum variant to future work (D2). A learned classifier or unverified middleware could target the same coordination outcome but supplies neither a machine-audited trust base nor a proved dominance guarantee.

LLM mathematics, autoformalization, and coordination. Our reliability motivation is sharpened by machine-generated formal mathematics: Wu et al. (2022) autoformalize natural-language statements into Isabelle/HOL with large language models, and benchmarks such as

miniF2F (Zheng et al., 2022) and *PutnamBench* (Tsoukalas et al., 2024) drive LLM theorem proving in Lean and other systems, where an unsound axiom or a stray `sorry` in a generated proof is the failure an axiom-footprint audit catches. On the behavioral side, Lorè & Heydari (2024) show that LLM cooperation in canonical games such as the Stag Hunt is swayed by contextual framing, not game structure alone, while Madmoun & Lahlou (2025) find that non-binding cheap talk lifts 4-player Stag-Hunt cooperation but offers no enforcement[2]; and Haupt et al. (2024) show binding contracts mitigate social dilemmas in multi-agent reinforcement learning with learned, not verified, policies.

We contribute a small, fully audited Lean 4 mechanization whose foundational-axiom footprint and single trust boundary are both machine-checked: a verifier-style component of the kind called for by the *Guaranteed Safe AI* agenda (Dalrymple et al., 2024). We argue that the same auditing practice transfers to verifying AI-generated mathematics.

3. The Lean-Verified Handshake

Game. We work over symmetric 2×2 games with action set $\{C, D\}$ (Cooperate / Defect) and payoff matrix entries $g_{cc}, g_{cd}, g_{dc}, g_{dd} \in \mathbb{Q}$, where g_{ab} is the row player's payoff when row plays a and column plays b. We focus on the *Stag-Hunt* family characterized by the ordering $g_{cd} < g_{dd} < g_{dc} < g_{cc}$: mutual cooperation is socially best; mutual defection is the safe fallback; the sucker's payoff (cooperate while the counterparty defects) is the worst outcome.

Asymmetric generalization. The formalization also covers *asymmetric Stag-Hunt* games, in which row and column may have distinct payoff matrices, each independently Stag-Hunt-ordered. We model this in Lean as an `AsymGame` structure carrying two `SymGame` records (row and column payoffs) and prove player-indexed counterparts of Theorems 4.1, 4.4, and Corollary 4.2 in `Asymmetric.lean`; a `SymGame.toAsym` coercion exhibits the symmetric development as the diagonal subcase. The mechanization is uniform precisely because each player's dominance argument depends only on that player's own Stag-Hunt chain, with no cross-player coupling in the proof body; per-player mirroring is therefore sound.

[2]In a preliminary pilot of our own (single model, Gemini 3.1 Flash-Lite; 4-player Stag-Hunt, payoffs $(10, 0, 8, 5)$), routing the same game through our *binding* `wrap` preserved full cooperation in every trial under a deceptive-broadcast framing where such non-binding cheap talk collapses, and the deceiver's stated reasoning matched the wrapper's machine-checked dominant strategy; the effect was robust to whether the prompt explained the binding rule. This is a single-shot, single-model observation, not a replication, and is orthogonal to this paper's axiom-audit methodology.

Wrapper. The handshake wrapper takes two attestations $a_A, a_B \in \{C, D\}$ (one per agent) and emits an action:

$$\mathrm{wrap}(a_A, a_B) = \begin{cases} C & \text{if } a_A = C \wedge a_B = C, \\ D & \text{otherwise.} \end{cases}$$

Both agents pass their attestations through wrap to obtain their physical actions. Agent A's action is $\mathrm{wrap}(a_A, a_B)$; agent B's action is $\mathrm{wrap}(a_B, a_A)$. Cooperation requires bilateral attestation; any attestation profile other than (C, C) collapses to mutual defection. By Lemma `wrap_comm`, the wrapper is symmetric in its arguments, so both agents always observe the same wrapped action and the wrapped meta-game's off-diagonal payoff cells (Table 1, right) are unreachable.

Table 1 compares the unconstrained game with the wrapped meta-game in the row player's payoff. Attesting C weakly dominates attesting D everywhere and strictly dominates when the counterparty attests C. A public bonus $\varepsilon > 0$ for attesting C closes the remaining tie when the counterparty attests D, rendering dominance strict everywhere (Corollary 4.3; deployment instantiations of ε are discussed under (L2) in §4).

Table 1. Row player's payoff g_{ab} in the Stag-Hunt game (left) and in the wrapped meta-game where both agents pass attestations a_A, a_B through the handshake (right). The wrapper collapses the off-diagonal cells to g_{dd}: any non-(C, C) profile yields the safe payoff (shaded).

Unconstrained		
	$b = C$	$b = D$
$a = C$	g_{cc}	g_{cd}
$a = D$	g_{dc}	g_{dd}

Wrapped		
	$a_B = C$	$a_B = D$
$a_A = C$	g_{cc}	g_{dd}
$a_A = D$	g_{dd}	g_{dd}

Trust model. The wrapper presupposes a *trusted attestation surface*: an external mechanism that publishes each player's attestation truthfully and unforgeably. We do *not* model this oracle as a Lean axiom. Instead, the trust hypothesis is mechanized as the `TrustedOracle` typeclass shown in Listing 1. A deployment instantiates the typeclass by exhibiting an attestation function and a truthfulness witness (relating the attested action to the player's underlying intent), after which the agent-level dominance theorems (`wrap_weakly_dominates_agent`, `wrap_pareto_unique_agent`, and Proposition 4.5) apply. The artifact ships a worked Lean instance, `deterministicChainOracle`, over a toy transaction model where `attest` forwards each transaction's intended action. Because the dominance theorem is oracle-agnostic through the typeclass, any other backend inherits the guarantee without re-proving the strategic core, provided it discharges the `truthful` obligation. Supplying that witness is the non-trivial, backend-specific burden; for TEE-

attestation and DID/VC backends we sketch it but do not yet mechanize it. Figure 1 shows the architecture.

4. Mechanized Theorem and Axiom-Footprint Audit

We mechanize the wrapper, a small commutativity lemma (`wrap_comm`: $\mathrm{wrap}(a,b) = \mathrm{wrap}(b,a)$, by case analysis on the two arguments) and its n-player counterpart `wrapN_comm`: $\mathrm{wrapN}(\mathrm{atts} \circ \sigma) = \mathrm{wrapN}(\mathrm{atts})$ for any $\sigma : \mathrm{Equiv.Perm}(\mathrm{Fin}\, n)$, the Stag-Hunt-ordering predicate, and the three main 2-player results (Theorems 4.1 and 4.4 together with Corollary 4.2) in Lean 4, using mathlib only for rational arithmetic, `linarith`, and finite-type support for the n-player development. The Stag-Hunt predicate is carried as a `Prop`-valued structure with three named fields, exposing the chain $g_{cd} < g_{dd} < g_{dc} < g_{cc}$ to `linarith`. The wrapper's commutativity makes the column player's weak-dominance claim coincide with the row player's; no separate column-player theorem is needed.

Theorem 4.1 (Weak dominance)**.** *For any `SymGame` g satisfying `StagHuntOrdering`, and any opponent attestation $oppAtt \in \{C, D\}$,*

$$\begin{aligned}&\mathrm{payoff}(g, \mathrm{wrap}(C, oppAtt), \mathrm{wrap}(oppAtt, C))\\ &\quad \geq \mathrm{payoff}(g, \mathrm{wrap}(D, oppAtt), \mathrm{wrap}(oppAtt, D)).\end{aligned}$$

Corollary 4.2 (Strict dominance under cooperation)**.** *When the opponent attests C, attesting C strictly beats attesting D:*

$$\begin{aligned}&\mathrm{payoff}(g, \mathrm{wrap}(C, C), \mathrm{wrap}(C, C))\\ &\quad > \mathrm{payoff}(g, \mathrm{wrap}(D, C), \mathrm{wrap}(C, D)).\end{aligned}$$

Corollary 4.3 (Strict dominance under ε-incentive)**.** *Let $\varepsilon > 0$ be a public per-round bonus added to the payoff of any agent who attests C (e.g., a regulatory subsidy, gas rebate, or reputation reward), independent of which wrapped action is realized. Then for any opponent attestation $oppAtt \in \{C, D\}$, attesting C* strictly *dominates attesting D in any Stag-Hunt-ordered game, including when the opponent does not commit ($oppAtt = D$), where weak dominance returns only equality:*

$$\begin{aligned}&\mathrm{payoff}(g, \mathrm{wrap}(C, oppAtt), \mathrm{wrap}(oppAtt, C)) + \varepsilon\\ &\quad > \mathrm{payoff}(g, \mathrm{wrap}(D, oppAtt), \mathrm{wrap}(oppAtt, D)).\end{aligned}$$

Mechanized as `wrap_strictly_dominates_eps` (2-player); the n-player, asymmetric, and `TrustedOracle` agent-level generalization is `wrapN_strictly_dominates_asym_eps`.

Theorem 4.4 (Pareto-uniqueness)**.** *For any attestation profile $(a, b) \neq (C, C)$,*

$$\begin{aligned}&\mathrm{payoff}(g, \mathrm{wrap}(C, C), \mathrm{wrap}(C, C))\\ &\quad > \mathrm{payoff}(g, \mathrm{wrap}(a, b), \mathrm{wrap}(b, a)).\end{aligned}$$

Proposition 4.5 (Cooperation realization)**.** *For any `TrustedOracle` instance and any agents a, b with $\mathrm{intent}(a) = \mathrm{intent}(b) = C$,*

$$\mathrm{wrap}(\mathrm{attest}(a), \mathrm{attest}(b)) = C.$$

The proofs are short: each dominance result reduces, after the wrapper's binary collapse, to a case split closed by `linarith` over a four-element payoff chain, a uniform template across the 2-player, n-player, symmetric, and asymmetric variants. Because the proofs are this simple, the whole development can be audited completely, a counterpoint to large opaque formalizations where axiom creep goes unnoticed: the contribution is *mechanization fidelity and the auditing methodology*, not proof depth. The non-uniform work sits in the supporting layer (permutation invariance of `wrapN`, the machine-checked specialization of the 2-player results from the n-player ones, the `Finset`-cardinality coercions behind the asymmetric lift), and the fidelity rests on parameterization over `StagHuntOrdering`, the typeclass-level separation of the trust assumption from the dominance proof (§3), and the build-time `#guard_msgs` assertions described below. Theorem 4.1 follows by case analysis on the action carried by $oppAtt$. In the $oppAtt{=}C$ branch, both sides reduce to g_{cc} and g_{dd} respectively, and the chain $g_{dd} < g_{dc} < g_{cc}$ closes the goal via `linarith`. In the $oppAtt{=}D$ branch, both sides collapse to g_{dd}. Corollary 4.2 sharpens the $oppAtt{=}C$ branch to strict inequality. Theorem 4.4 cases on a and b; the excluded (C, C) case is contradicted by hypothesis, the others reduce to $g_{cc} > g_{dd}$. Proposition 4.5 closes by structural rewrite from the `truthful` field; this is the only result that productively exercises the typeclass, and is consequently axiom-free in the audit below. The same case-split-then-`linarith` template carries to the asymmetric counterparts in `Asymmetric.lean` (`wrap_*_asym_{row,col}`). Listing 2 shows the central theorem.

n-player generalization. The same template extends to $n \geq 2$ players. The wrapper wrapN outputs C iff every attestation carries C, so the wrapped meta-game collapses to two reachable outcomes, reducing the dominance argument to their two payoffs: the all-C payoff $g.u\,C\,(n{-}1)$ (the top of the four-step chain $g.u\,C\,0 < g.u\,D\,0 < g.u\,D\,(n{-}1) < g.u\,C\,(n{-}1)$ encoded in `StagHuntOrderingN`) and the all-D payoff $g.u\,D\,0$ (the chain's second element). We formalize this as `SymGameN n`, state weak dominance as Theorem 4.6 and Pareto-uniqueness as Theorem 4.7, and prove the truthful-oracle result `cooperate_realized_n_truthful` (axiom-free); details in Appendix A. The asymmetric case mirrors the per-player framework formalized in `AsymStagHuntOrderingN`, with

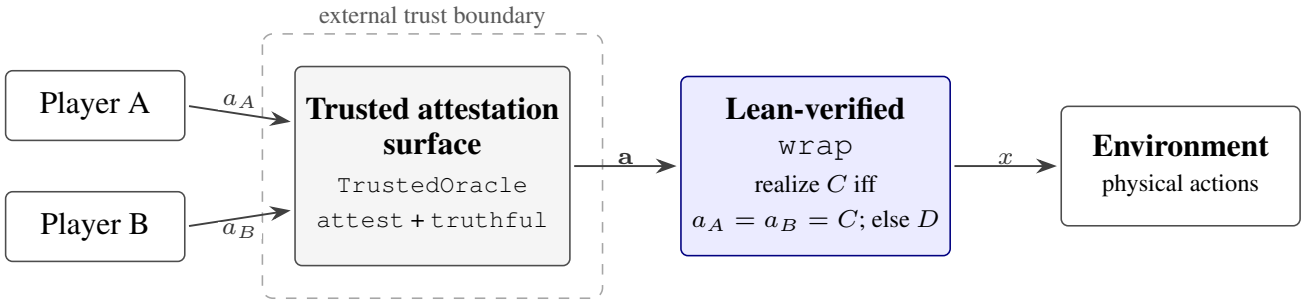

Figure 1. Architecture of the verified handshake. Participating players submit attestations to an external trusted attestation surface, formalized by `TrustedOracle`; the dashed box marks the external trust boundary, a deployment-surface assumption, not a Lean axiom. The Lean-verified `wrap` combinator realizes C iff both attestations are C and realizes D otherwise; the realized action $x \in \{C, D\}$ is played in the environment. Shown for $n = 2$; for arbitrary n, the profile $\mathbf{a} = (a_i)_{i=1}^n$ is routed to `wrapN`, which realizes C iff all $a_i = C$.

Listing 1. The mechanized trust-model typeclass. A Lean typeclass is an interface a deployment must implement: here, an `attest` function (mapping each agent to its attestation) and a `truthful` witness (a proof that the attested action equals the agent's intent). Agent-level dominance theorems take agents as inputs, with the typeclass supplying their attestations; the typeclass is agent-count-agnostic, lifting unchanged to the n-player setting (Theorem 4.6).

```
class TrustedOracle (Agent : Type)
    (intent : Agent → Action) where
  attest : Agent → Attestation
  truthful :
    ∀ a, (attest a).action = intent a
```

`wrapN_weakly_dominates_asym` proved analogously.

Theorem 4.6 (Weak dominance, n-player)**.** *For any $n \geq 2$, any `SymGameN` g satisfying `StagHuntOrderingN`, any player i, and any opponents' attestations a_{-i},*

$$\begin{aligned}&\mathrm{payoffN}_g\big(\mathrm{wrapN}(a_{-i}, C)\big)\\ &\quad \geq \mathrm{payoffN}_g\big(\mathrm{wrapN}(a_{-i}, D)\big).\end{aligned}$$

Theorem 4.7 (Pareto-uniqueness, n-player)**.** *For any attestation profile $a \neq (C, \ldots, C)$, $\mathrm{payoffN}_g(\mathrm{wrapN}((C, \ldots, C))) > \mathrm{payoffN}_g(\mathrm{wrapN}(a))$.*

Unification. The 2-player development (Theorems 4.1, 4.4, Corollary 4.2, and their asymmetric mirrors) is formally a specialization of the n-player development at $n = 2$: diagonal lifts (`SymGame.toSymGameN`, `AsymGame.toAsymGameN`) embed the 2-player game types, preservation lemmas carry the Stag-Hunt chain through, and bridge lemmas (`payoffN_two_eq_payoff` and its asymmetric analogues) collapse the n-player wrapped payoffs to their 2-player forms, so the `wrap_*_via_n` corollaries (three symmetric, six asymmetric) obtain the 2-player theorems by instantiating their n-player counterparts at $n = 2$. The parallel APIs are thus connected by machine-checked specialization on both the symmetric and asymmetric branches.

4.1. Axiom Hygiene as a Regression Test

`#print axioms` (which lists a declaration's axiom dependencies) and `#guard_msgs` (which fails elaboration unless a command's output matches an expected string) are both standard Lean facilities. Our contribution is to compose them into a *per-declaration regression test*: after each theorem we assert its expected foundational-axiom footprint with `#guard_msgs in #print axioms`, so any drift fails the build with a mismatch (`Audit.lean`; full source at (Ondras, 2026)). The check is prover-agnostic: `#guard_msgs in #print axioms` is the Lean-specific mechanism, but pinning each declaration's axiom footprint and rejecting any drift in CI transfers to any system whose kernel exposes axiom dependencies (e.g. Rocq's `Print Assumptions` or Isabelle's theorem-dependency tracking), where the gate is a small CI wrapper. It also scales cheaply: the audit adds one independent assertion per declaration (parallelizable across a build), and these assertions can be auto-generated by snapshotting a known-good build rather than written by hand, so the method applies to large developments as easily as to this small one. The development is correspondingly small: its 2-player core imports *mathlib* only for $\mathbb{Q}$-arithmetic and `linarith`, with the n-player and asymmetric layers additionally drawing on finite-type support (`Fintype`, `Finset`, `Fin`) and the `norm_num`/`fin_cases` tactics; it has *no* `sorry`s and *no* custom axioms, builds in ∼30 s on a warm cache (Lean 4 `v4.29.1`, *mathlib* `v4.29.1`), and gates its trust base in CI. Every one of its 56 declarations depends only on a subset of Lean 4's foundational set `[propext, Classical.choice, Quot.sound]` (kernel+Classical); Table 2 partitions them by footprint. Most declarations lie at the extremes: characterization lemmas and the truthful-oracle results `cooperate_realized_*_truthful` are axiom-free

Listing 2. The mechanized weak-dominance theorem. The proof has two cases on the opponent's attestation: when it is `C`, both wrapped sides reduce to `g.cc` and `g.dd`, and the goal $g_{cc} \geq g_{dd}$ follows from the Stag-Hunt chain $g_{dd} < g_{dc} < g_{cc}$; when it is `D`, both sides collapse to `g.dd`. In Lean: `obtain` unpacks the attestation; `cases` splits on the underlying action; `simp` unfolds `wrap` and `payoff`; `linarith` closes via the chain.

```
theorem wrap_weakly_dominates
    (g : SymGame) (h : StagHuntOrdering g) (oppAtt : Attestation) :
    payoff g (wrap ⟨.C⟩ oppAtt) (wrap oppAtt ⟨.C⟩)
      ≥ payoff g (wrap ⟨.D⟩ oppAtt) (wrap oppAtt ⟨.D⟩) := by
  obtain ⟨a⟩ := oppAtt
  cases a
  · -- a = .C: g.cc ≥ g.dd via g.dd < g.dc < g.cc
    simp [wrap, payoff]
    linarith [h.dd_lt_dc, h.dc_lt_cc]
  · -- a = .D: g.dd ≥ g.dd
    simp [wrap, payoff]
```

(pure computation or structural rewrite), while results invoking `simp`/`linarith` pull the full kernel+Classical set; only a handful fall in between (`propext` from `Iff` rewrites, `Quot.sound` from `Finset` permutation reasoning). The full per-declaration breakdown is in Appendix B.

Scope and limitations. We record nine explicit limitations. (L1) The cooperation-realization guarantee relies on the `truthful` field of `TrustedOracle`: a non-truthful witness collapses it. The artifact ships `deterministicChainOracle`, a `TrustedOracle` instance over a richer agent type (`Transaction`, carrying a sender alongside the intended action) with truthfulness discharged by structure projection on the assumption that the chain forwards intent faithfully, alongside a `lyingAttest` non-instance (`lyingAttest_not_truthful`) witnessing that the obligation is non-vacuous. In scope is exactly one obligation: a single oracle that attests truthfully and unforgeably; out of scope, deferred to the deployment surface, are spoofed or false attestations, wrapper bypass, replay, cross-counterparty equivocation, signer collusion, compromised TEEs, validator capture, Sybil attacks, and post-attestation policy changes (TEE and DID/VC instances and adversarial-oracle robustness are likewise unmodelled). Any of these voids the guarantee; the dominance ordering is proved oracle-agnostically. (L2) Strict dominance follows from the weak result via an external ε-incentive (Corollary 4.3): a public bonus $\varepsilon > 0$ for attesting C closes the tie when the opponent does not commit. Instantiations of ε (for example, a regulatory subsidy) are governance-layer choices deferred to deployment, and the strategic core of the wrapper remains independent of ε's source. Tremble-robust strict variants *without* external incentive (cf. Sauerberg & Oesterheld (2026)'s ex post verifiable commitments) remain follow-up work. (L3) The wrapper's binary collapse to all-C / all-D means the mechanized theorems cover partial-attestation profiles ($k < n$ players commit) uniformly but trivially: any single defection forces the all-defector payoff regardless of the other attestations. A richer payoff structure supporting *threshold cooperation* (intermediate counts giving intermediate payoffs) would make partial adoption strategically meaningful and is deferred; see (D1) for follow-up. (L4) Strategies are pure; mixed-strategy and behavioral-strategy variants are not yet mechanized. (L5) The dominance theorem assumes all participating agents route through the wrapper. Adoption itself is not an equilibrium concept of the model: under purely voluntary adoption, the choice to wrap has Stag-Hunt structure. We treat the closure of this meta-game as a deployment/governance concern (§5), not a formal-verification concern. (L6) The asymmetric extensions (`Asymmetric.lean` for the 2-player case; `NPlayerAsym.lean` for the n-player case) require *every* player's payoff structure to be Stag-Hunt-ordered at the relevant endpoint profiles; hybrid-ordering games (e.g., one player Stag-Hunt while another follows Prisoner's Dilemma) are out of scope. (L7) Each wrapper invocation requires one attestation per agent through the chosen backend, so per-round cost scales with the number of agents and the oracle's per-attestation latency (chain consensus for smart contracts, remote-attestation handshake for TEEs, credential resolution for DIDs/VCs), aggregating across rounds; these backend-specific factors lie outside our single-shot formal model. (L8) The three oracle backends also differ on revocation and jurisdictional exposure: smart contracts have no native revocation, TEEs revoke per enclave instance, and DIDs/VCs revoke per credential, with jurisdiction following the chain, the hardware vendor, or the credential issuer respectively. These dimensions lie outside our formal model and are left to the deployment layer. (L9) The two methods are demonstrated on a single small (56-declaration) development. The axiom-hygiene gate scales *mechanically* to large formalizations (its per-declaration cost is local and its assertions can be auto-generated; §4.1), but we give no *empirical* evidence

Table 2. Foundational-axiom footprint of all 56 audited declarations, summarizing the build-time `#guard_msgs in #print axioms` guards in `Audit.lean`. For the complete per-declaration breakdown, see Table 3.

Axiom footprint	# declarations
axiom-free	8
`propext` only	5
`Quot.sound` only	1
`propext` + `Quot.sound`	1
`propext` + `Classical.choice` + `Quot.sound`	41
Total	**56**

at that scale, precisely the regime where unnoticed axiom creep is the motivating risk, nor that the trust-boundary pattern generalizes beyond this one mechanism; transfer to AI-generated proofs is argued, not yet shown (§5, D3).

5. Discussion and Outlook

What we verify here is a strategic combinator and its trust boundary, not a general AI-alignment solution: a small, fully audited Lean 4 development whose foundational-axiom footprint and single unproved assumption are both machine-auditable. The result is independent of agent internals; the surrounding middleware (oracle, transport, signing) and its trust profile are deferred to the chosen deployment surface, and the strategic core is provably independent of it.

Relevance to verifying AI-generated mathematics. These contributions relate to two research directions. In verification and evaluation, axiom-footprint auditing is agnostic to whether a proof was written by a human or generated by an LLM, so the same `#guard_msgs` gate that protects our development would serve as a model-independent reliability check for autoformalized or AI-proved mathematics, catching a stray `sorry` or unexpected axiom in a generated proof regardless of its source (demonstrated here only on hand-written proofs). This complements the faithfulness focus of autoformalization (Wu et al., 2022) and benchmarks such as *miniF2F* (Zheng et al., 2022) and *PutnamBench* (Tsoukalas et al., 2024). Toward self-evolving scientific agents, a per-theorem axiom-hygiene gate guards against an agent "proving" its own conjecture through an unsound axiom it introduced. The case study also touches formal methods in science (it formalizes a mechanism-design result) and human-AI collaboration (the typeclass trust boundary makes explicit, for a human auditor, exactly which step is proved and which is assumed).

From a verified kernel to deployment. The wrapper presupposes that all parties have *adopted* it; under voluntary adoption that choice is itself a Stag Hunt (full adoption yields g_{cc}, partial adoption falls back to the original game), so the mechanism relocates coordination from per-action play to a one-time, publicly verifiable adoption decision rather than eliminating it. Closing that residual gap is a deployment-and-governance question outside our formal model, as are the choice of attestation backend (smart contract, TEE, or DID/VC) and its revocation and jurisdictional profile.

We outline three follow-up directions. **(D1) Partial-attestation thresholds**: when only k of n parties commit, the binary collapse to all-C/all-D no longer applies; a richer payoff function would yield a *threshold cooperation* theorem (cooperate if at least k^* of n commit), extending the count-indexed `SymGameN n` already in the development, the one extension that yields a non-trivial inequality. **(D2) Byzantine attestation oracles**: the `TrustedOracle` typeclass extends to a quorum oracle tolerating up to f Byzantine signers (e.g., a $\lceil n/3 \rceil + 1$ safe threshold), building on the present `deterministicChainOracle`; no weaker-than-truthful guarantee is presently mechanized. **(D3) Verifying AI-generated proofs**: the immediate next step is to run the axiom-hygiene auditor on a batch of machine-generated Lean proofs (LLM-autoformalized statements of our own theorems together with existing model-generated proofs from benchmarks such as *miniF2F* (Zheng et al., 2022), across several models and tens of proofs per model), reporting `sorry`, unexpected-axiom, and faithfulness rates alongside the build failures the gate raises. This would turn the methodology from one demonstrated on hand-written proofs into one validated on machine-generated ones.

Formal verification has long examined whole systems such as compilers, kernels, and cryptographic primitives; the two methods here apply that scrutiny to the axioms and the single trust assumption a proof itself rests on.

Software and Data

The Lean 4 source and all machine-checked proofs, including the axiom-footprint audit (`Audit.lean`), are available at https://github.com/jancio/VerifiedHandshake.

Impact Statement

This paper contributes methods for *trustworthy formalization*: auditing which foundational axioms a proof depends on, and isolating where an unproved assumption enters. The intended impact is on the reliability of machine-checked and, increasingly, machine-*generated* mathematics: an executable axiom-footprint audit lets authors, reviewers, and downstream users see what a formal result actually rests on, reducing the risk of over-trusting a proof that type-checks but silently depends on a strong axiom or a `sorry`. The case-study object is a cooperation-inducing commitment mechanism; should such mechanisms be deployed to coordinate AI agents, the same trust-boundary approach is intended to make a deployment's assumptions visible to auditors rather than to advance any particular deployment as universally trustworthy. Binding commitment devices can also entrench power asymmetries if the trust surface is captured by a few validators, and the kernel does not itself address fairness, consent, or representation.

References

Bagnall, A., Merten, S., and Stewart, G. A library for algorithmic game theory in Ssreflect/Coq. *Journal of Formalized Reasoning*, 10(1):67–95, 2017. doi: 10.6092/issn.1972-5787/7235.

Barász, M., Christiano, P., Fallenstein, B., Herreshoff, M., LaVictoire, P., and Yudkowsky, E. Robust cooperation in the prisoner's dilemma: Program equilibrium via provability logic. *arXiv preprint arXiv:1401.5577*, 2014.

Caminati, M. B., Kerber, M., Lange, C., and Rowat, C. Sound auction specification and implementation. In *Proceedings of the Sixteenth ACM Conference on Economics and Computation (EC '15)*, pp. 547–564, 2015. doi: 10.1145/2764468.2764511.

Critch, A. A parametric, resource-bounded generalization of Löb's theorem, and a robust cooperation criterion for open-source game theory. *The Journal of Symbolic Logic*, 84(4):1368–1381, 2019. doi: 10.1017/jsl.2017.42.

Dalrymple, D., Skalse, J., Bengio, Y., Russell, S., Tegmark, M., Seshia, S., Omohundro, S., Szegedy, C., Goldhaber, B., Ammann, N., et al. Towards guaranteed safe AI: A framework for ensuring robust and reliable AI systems. *arXiv preprint arXiv:2405.06624*, 2024.

de Moura, L. and Ullrich, S. The Lean 4 theorem prover and programming language. In *Automated Deduction – CADE 28*, volume 12699 of *Lecture Notes in Computer Science*, pp. 625–635. Springer, 2021. doi: 10.1007/978-3-030-79876-5_37.

Haupt, A. A., Christoffersen, P. J. K., Damani, M., and Hadfield-Menell, D. Formal contracts mitigate social dilemmas in multi-agent reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 38:51, 2024. doi: 10.1007/s10458-024-09682-5.

Kalai, A. T., Kalai, E., Lehrer, E., and Samet, D. A commitment folk theorem. *Games and Economic Behavior*, 69 (1):127–137, 2010.

Le Roux, S., Martin-Dorel, É., and Smaus, J.-G. An existence theorem of Nash equilibrium in Coq and Isabelle. In *Proceedings of the 8th International Symposium on Games, Automata, Logics, and Formal Verification (GandALF 2017)*, volume 256 of *EPTCS*, pp. 46–60, 2017. doi: 10.4204/EPTCS.256.4.

Lorè, N. and Heydari, B. Strategic behavior of large language models and the role of game structure versus contextual framing. *Scientific Reports*, 14:18490, 2024. doi: 10.1038/s41598-024-69032-z.

Madmoun, H. and Lahlou, S. Communication enables cooperation in LLM agents: A comparison with curriculum-based approaches. *arXiv preprint arXiv:2510.05748*, 2025. arXiv v3 dated 11 Mar 2026; EACL 2026.

MixedMatched contributors. Formalizing game theory in Lean 4. https://github.com/MixedMatched/formalizing-game-theory, 2024.

Oesterheld, C. Robust program equilibrium. *Theory and Decision*, 86(1):143–159, 2019. doi: 10.1007/s11238-018-9679-3.

Ondras, J. Verified Handshake: Lean 4 Source Code and Machine-Checked Proofs. https://github.com/jancio/VerifiedHandshake, 2026.

Pomeret-Coquot, P., Fargier, H., and Martin-Dorel, É. BelGames: A formal theory of games of incomplete information based on belief functions in the Coq proof assistant. In *14th International Conference on Interactive Theorem Proving (ITP 2023)*, volume 268 of *LIPIcs*, pp. 25:1–25:19, 2023. doi: 10.4230/LIPIcs.ITP.2023.25.

Sauerberg, N. and Oesterheld, C. Promises made, promises kept: Safe Pareto improvements via ex post verifiable commitments. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(20):17231–17241, 2026. Oral presentation; arXiv:2505.00783.

Tennenholtz, M. Program equilibrium. *Games and Economic Behavior*, 49(2):363–373, 2004.

The mathlib Community. The Lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2020)*, 2020. doi: 10.1145/3372885.3373824.

Tsoukalas, G., Lee, J., Jennings, J., Xin, J., Ding, M., Jennings, M., Thakur, A., and Chaudhuri, S. PutnamBench: Evaluating neural theorem-provers on the Putnam mathematical competition. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2024. arXiv:2407.11214.

Wu, Y., Jiang, A. Q., Li, W., Rabe, M. N., Staats, C., Jamnik, M., and Szegedy, C. Autoformalization with large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. arXiv:2205.12615.

Zheng, K., Han, J. M., and Polu, S. miniF2F: A cross-system benchmark for formal Olympiad-level mathematics. In *International Conference on Learning Representations (ICLR)*, 2022. arXiv:2109.00110.

A. n-Player Generalization (Lean Source)

We list the central definitions and theorem signatures (Listing 3); full proofs are at (Ondras, 2026).

Symmetric case. From `NPlayer.lean`:

Listing 3. The mechanized n-player generalization (symmetric case). Shown are the count-indexed game `SymGameN`, the n-ary wrapper `wrapN` (realizing C iff every attestation carries C), and the two main theorem signatures; their full statements are Theorems 4.6–4.7, with full proofs in the artifact.

```
structure SymGameN (n : ℕ) where
  u : Action → Fin n → ℚ
def wrapN {n : ℕ} (atts : Fin n → Attestation) : Action :=
  if ∀ i, (atts i).action = .C then .C else .D
theorem wrapN_weakly_dominates {n : ℕ} (hn : 2 ≤ n) ... -- weak dominance (Thm 4.6)
theorem wrapN_pareto_unique     {n : ℕ} (hn : 2 ≤ n) ... -- Pareto-uniqueness (Thm 4.7)
```

Asymmetric case. Mirror with `AsymGameN n` and the per-player `wrapN_weakly_dominates_asym`; the wrapper is unchanged since it is action-agnostic. A coercion `symGameN_toAsym` embeds every symmetric n-player game in the asymmetric framework, with the lifting lemma `stagHuntN_toAsym` closing the obligation that a symmetric Stag-Hunt remains Stag-Hunt under the embedding.

Truthful-oracle capstone. The agent-level lift `cooperate_realized_n_truthful` is axiom-free: an inline `unfold wrapN; if_pos hAll` discharged by `TrustedOracle.truthful`, deliberately avoiding rewriting through `wrapN_eq_C_iff` (the lemma is itself axiom-free, but the rewrite tactic for an `Iff` would introduce `propext` into the closing proof's transitive dependencies).

B. Per-Declaration Axiom Audit

Table 2 gives the trust base by footprint; Table 3 lists every audited declaration with its footprint. Beyond the foundational set `[propext, Classical.choice, Quot.sound]`, the development appeals to no mathlib lemmas other than decidable equality on `Action`, finite-type universal-quantifier decidability and `Finset`-cardinality reasoning for the n-player and asymmetric lifts, and `linarith`'s internal use of $\mathbb{Q}$-linear arithmetic; there are no `sorry`s and no custom axioms.

Table 3. Per-declaration foundational-axiom footprint of all 56 audited declarations, read off the build-time `#guard_msgs in #print axioms` guards in `Audit.lean` (the detail behind Table 2). A check (✓) marks an axiom a declaration depends on: `P=propext`, `C=Classical.choice`, `Q=Quot.sound`; an all-blank row is axiom-free. The *Group* column groups rows by footprint tier (whose sizes match Table 2) and, within the full-kernel tier, by structural family.

Group	Declaration	File (.lean)	P	C	Q
axiom-free	wrap_comm	Theorem			
	wrapN_eq_C_iff	NPlayer			
	cooperate_realized_under_truthful_oracle	TrustedOracle			
	cooperate_realized_n_truthful	NPlayer			
	lyingAttest_not_truthful	Deployments			
	lyingAttest_n_not_truthful	Deployments			
	stagHunt_toAsym	Asymmetric			
	SymGame.toSymGameN	NPlayer			
propext	wrap_eq_C_iff	Theorem	✓		
	wrapN_two_eq_wrap	NPlayer	✓		
	AsymGame.toAsymGameN	NPlayerAsym	✓		
	asymPayoffN_two_eq_asymPayoffRow	NPlayerAsym	✓		
	asymPayoffN_two_eq_asymPayoffCol	NPlayerAsym	✓		
Quot.sound	wrapN_comm	NPlayer			✓
propext + Quot.sound	payoffN_two_eq_payoff	NPlayer	✓		✓
2-player	wrap_weakly_dominates	Theorem	✓	✓	✓
	wrap_strict_when_opponent_commits	Theorem	✓	✓	✓
	wrap_pareto_unique	Theorem	✓	✓	✓
	wrap_weakly_dominates_agent	TrustedOracle	✓	✓	✓
	wrap_pareto_unique_agent	TrustedOracle	✓	✓	✓
2-player asym.	wrap_weakly_dominates_asym_row	Asymmetric	✓	✓	✓
	wrap_weakly_dominates_asym_col	Asymmetric	✓	✓	✓
	wrap_strict_when_opponent_commits_asym_row	Asymmetric	✓	✓	✓
	wrap_strict_when_opponent_commits_asym_col	Asymmetric	✓	✓	✓
	wrap_pareto_unique_asym_row	Asymmetric	✓	✓	✓
	wrap_pareto_unique_asym_col	Asymmetric	✓	✓	✓
n-player	wrapN_weakly_dominates	NPlayer	✓	✓	✓
	wrapN_strict_when_others_commit	NPlayer	✓	✓	✓
	wrapN_pareto_unique	NPlayer	✓	✓	✓
	wrapN_weak_nash	NPlayer	✓	✓	✓
	wrapN_weakly_dominates_agent	NPlayer	✓	✓	✓
	wrapN_pareto_unique_agent	NPlayer	✓	✓	✓
n-player asym.	wrapN_weakly_dominates_asym	NPlayerAsym	✓	✓	✓
	wrapN_strict_when_others_commit_asym	NPlayerAsym	✓	✓	✓
	wrapN_pareto_unique_asym	NPlayerAsym	✓	✓	✓
	wrapN_weakly_dominates_asym_agent	NPlayerAsym	✓	✓	✓
	wrapN_pareto_unique_asym_agent	NPlayerAsym	✓	✓	✓
lifts	stagHuntN_toAsym	NPlayerAsym	✓	✓	✓
	stagHuntOrdering_toN	NPlayer	✓	✓	✓
	asymStagHuntOrdering_toN	NPlayerAsym	✓	✓	✓
specialization	wrap_weakly_dominates_via_n	NPlayer	✓	✓	✓
	wrap_strict_when_opponent_commits_via_n	NPlayer	✓	✓	✓
	wrap_pareto_unique_via_n	NPlayer	✓	✓	✓
	wrap_weakly_dominates_asym_row_via_n	NPlayerAsym	✓	✓	✓
	wrap_weakly_dominates_asym_col_via_n	NPlayerAsym	✓	✓	✓
	wrap_strict_when_opponent_commits_asym_row_via_n	NPlayerAsym	✓	✓	✓
	wrap_strict_when_opponent_commits_asym_col_via_n	NPlayerAsym	✓	✓	✓
	wrap_pareto_unique_asym_row_via_n	NPlayerAsym	✓	✓	✓
	wrap_pareto_unique_asym_col_via_n	NPlayerAsym	✓	✓	✓
ε-incentive	wrapN_strictly_dominates_asym_eps	Incentive	✓	✓	✓
	wrapN_strictly_dominates_asym_eps_agent	Incentive	✓	✓	✓
	wrapN_strictly_dominates_eps	Incentive	✓	✓	✓
	wrapN_strictly_dominates_eps_agent	Incentive	✓	✓	✓
	wrap_strictly_dominates_eps	Incentive	✓	✓	✓
	wrap_strictly_dominates_asym_row_eps	Incentive	✓	✓	✓
	wrap_strictly_dominates_asym_col_eps	Incentive	✓	✓	✓