

Catching collusion in useful-work verification

Oct 25, 2026

Aeriella Goulden

aeriella@annotrain.org

--

An entry for the Apart Research AI Collusion Research Sprint, 23 to 25 October 2026.

Abstract

Systems that pay for access with useful work often have participants check each other's work, which invites collusion between workers and checkers. That includes AI agents that share a model. We ask which single restriction removes the most collusion at the least cost (question 1.5), in a peer-verified useful-work turnstile that guards a network relay. Our seeded simulation has 200 workers, 12 verifier nodes, 50 audited epochs and 30 seeds per condition. A cartel controls a share of both roles, against five switchable defences: commit-reveal assignment of workers and of verifier panels, blinding, hidden canaries, and planted faults (honeypots).

The controls behave as expected. With no cartel, no wrong answer is accepted. With a 20% cartel and no defences, $20.1\% \pm 0.5\%$ are. The restrictions interact strongly. Redundancy alone makes collusion worse (49.2% of wrong answers accepted), because colluders steer partners onto the same items. Commit-reveal alone does nothing. Together they reach 0.75%. Canaries are the cheapest single restriction (7.8% at 1.18 answers per item), and the only one that removes a cartel that lies only when unanimous.

On the verifier side, random panels are the best single restriction (0.06%, at no added cost). Blinding or planted faults alone do nothing, but together they expose every colluding verifier. A cautious cartel that lies on 5% of its chances causes 0.13% wrong answers and is almost never caught. Audits bound the damage, but they don't remove patient colluders.

Introduction

More and more systems pay for access, or reward participants, with work that other participants check: proof-of-useful-work, crowdsourced labelling, decentralised compute, and agent marketplaces where AI agents do tasks and other agents grade them.

In every one of these, the workers and the checkers can collude. If AI agents fill both roles, they may coordinate without anyone designing it, through shared training, shared prompts, or a side channel.

Our concrete case is a mailbox relay (/mbx) that carries WireGuard packets between devices on a mesh network. A security audit found that any device holding a valid capability can flood a mailbox and burn the relay's storage quota. The fix we propose is a turnstile: before a device gets more sending budget, it answers an opaque ticket of useful work. The work might be a compute slice, a fetch of a public resource that needs refreshing, a retrieval proof for a pinned file, or a check of someone else's work. The answers are then checked in batches by peer nodes, who must not learn whose work it is, or anything private about it.

This entry asks the sprint's question 1.5 outside a market: **which single restriction removes the most collusion at the least cost?** We measure cost as the honest effort each restriction adds per real item: extra answers plus verifier checks. Two of our restrictions are also instances of question 3.2's trusted-agent idea. **Canaries** are work items whose answer the system already knows, and **planted faults** are fake disagreements that a verifier must flag. Both are honeypots that a colluder can't tell from real work.

Related work

Collusion between AI agents. LLM pricing agents collude in markets, and innocuous prompt cues increase it (Fish et al. 2024). Private channels between pairs of agents create coalition advantage in workplace tasks (Nakamura et al. 2026). Models from similar families coordinate better (Aharon et al. 2026), two deployments of the same base model show small correlations where different base models show none (Xu et al. 2026), and the family behind a text can be identified from its style (Dietrich 2026). That is why we treat a cartel that coordinates only by making the same choice, with no messages, as a realistic case.

Single restrictions. Earlier work measured one lever at a time: raising sampling temperature (Xu et al. 2026), throttling messages (Agrawal et al. 2025), mixing in non-LLM agents (Keppo et al. 2026), a single no-regret participant (Collina et al. 2025), and a refundable deposit forfeited if caught (Q. Liu et al. 2026). We add verification-side levers (assignment, blinding, redundancy, honeypots) and measure them both alone and together, with their cost.

Honeypots and planted attacks. Untrusted monitors have been tested on planted attacks (Greenblatt et al. 2023), and honeypot tasks have exposed colluding agents (Tao et al. 2026). Our canaries and planted faults apply the same idea to peer verification, where the planted item has to be indistinguishable from real work for it to work at all.

What is reused. The design draws on our own earlier DHF / ECCA work, which predates the sprint: decaying capacity tokens and a non-decaying ResidueToken earned by repairing faults (DHF Tokenomics), and the Playfair test's per-epoch fairness audit (DHF Playfair). The attestation function `createPlayfairAttestation` and a SHA-256 helper are pre-existing code. The simulation, its experiments and this report were written during the sprint. No sprint starter code was used.

Methods

The simulation (`src/lib/turnstile/collusion-sim.ts`) is pure and seeded: the same configuration and seed always produce the same run. Each experiment averages 30 seeds.

Population. There are 200 workers and 12 verifier nodes. The first share of each (the colluder share) is in one cartel. Each epoch has 120 real items. There are 50 epochs, matching Playfair.

Each epoch.

1. Every item gets a hidden true answer and goes to $r = 3$ distinct active workers.
Canaries are extra items with known answers (15% of tickets by default), assigned exactly like real ones.
2. An honest worker answers correctly, except with an error rate of 0.2%. A cartel member answers with the cartel's agreed wrong value when the cartel strategy says to lie.
3. A wrong answer to a canary is a strike, checked inline.
4. An item whose answers all agree is accepted. Items whose answers disagree go into batches of 64. Each batch is checked by 3 verifiers and carries 2 **planted faults**: fake disagreements that each verifier must flag.
5. A disagreement flagged by any verifier is recomputed by the node. This opens a residue, worth 5 ReT in DHF terms, and every wrong answer is a strike. An unflagged disagreement is settled by majority. A verifier that misses a planted fault gets a strike.
6. Two strikes revoke a worker's or verifier's capability.
7. **Audit**: the epoch is FAIR only if no wrong answer was accepted. The epoch's audit is chained into a Playfair attestation, a hash commitment linked to the previous epoch's.

Cartel strategies.

- **Unanimous**: lie only on items where every assignee is in the cartel. Members find out by sharing ticket ids over a side channel. No honest answer disagrees, so nothing goes to the verifiers.

- **Majority:** lie whenever the cartel holds a majority of an item's assignees, and rely on cartel verifiers to suppress the disagreement.
- **Caution:** the cartel lies on only a fraction `lieProb` of the items it could lie on.

Defences, each switchable for ablation:

Defence	On	Off (what the cartel can do)
Commit-reveal assignment (workers)	Assignment comes from a seed committed before answers arrive	A member that draws an item pulls partners onto it
Commit-reveal panels (verifiers)	Panels are drawn the same way	Cartel verifiers volunteer for batches holding the cartel's lies
Blinding	Verifiers see per-batch blinded tags and digests	Verifiers know which disagreements are the cartel's
Canaries	Known-answer items, indistinguishable from real ones	None
Planted faults	Fake disagreements in every batch	None

A blinded cartel verifier can't tell which disagreements, or which batches, belong to the cartel. So its only way to help is to suppress whole batches at a guess, with probability `verifierSuppressProb`.

Metrics: the share of real items where a wrong answer was accepted; the share of cartel workers revoked; the share of honest workers revoked; cartel verifiers revoked; fair epochs out of 50; answers computed per real item (cost); and the epoch of first detection.

Cost is honest effort per real item: worker answers (redundancy and canaries) plus verifier checks (disputes and planted faults, times the panel size). A system with no defences costs 1.00.

Controls. A negative control (no cartel, all defences) checks that defences don't flag honest work as collusion, apart from the honest error rate. A positive control (a 20% cartel, no defences) checks that collusion succeeds when nothing stops it. **Experiments:** E1 to E4 vary the cartel's size, the canary share, redundancy and the cartel's caution. E5 and E6 remove one defence at a time from the full set (leave-one-out). E7 and E8 add one defence at a time to none (single restrictions, question 1.5). Every table reports the mean and standard deviation over 30 seeds, and `npx tsx scripts/turnstile-collusion.ts` reproduces every number in about 10 s on a laptop, with no GPU and no API calls.

Results

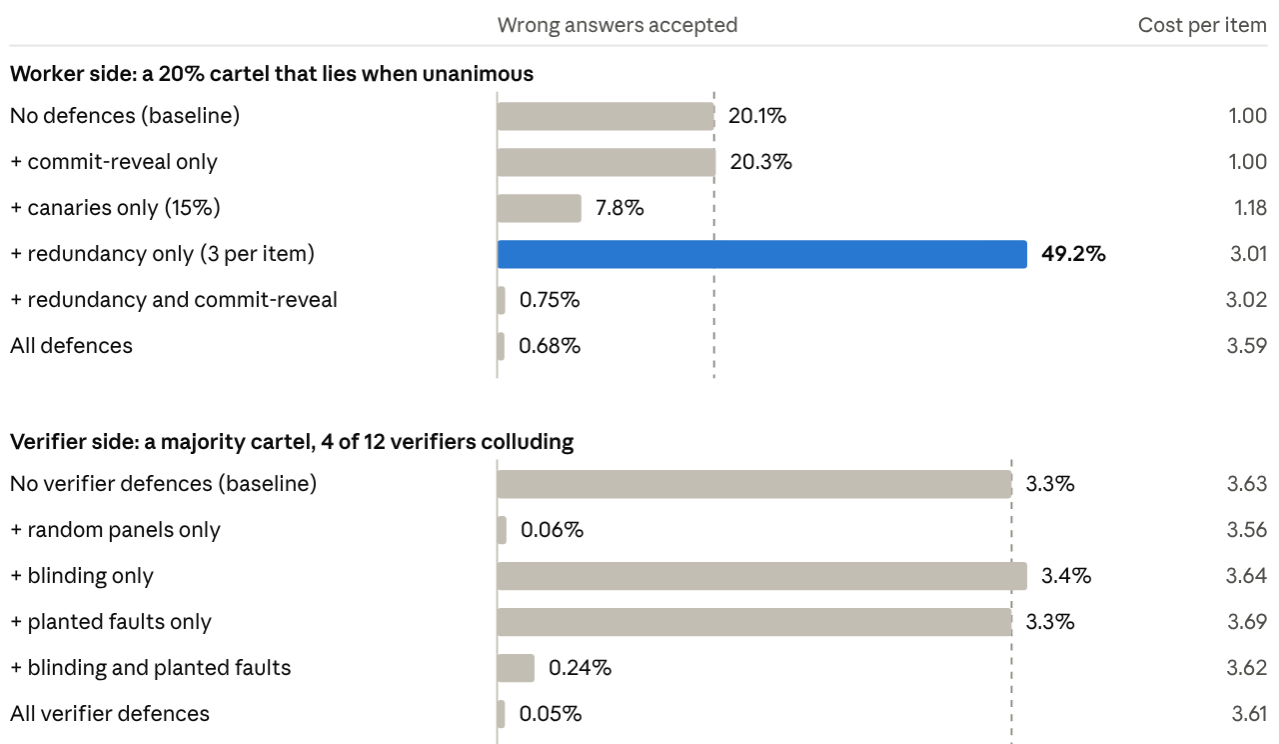
Controls

Both controls behave as intended. With no cartel and every defence on, no wrong answer was accepted in any of the 30 runs ($0.00\% \pm 0.00\%$), all 50 epochs were fair, and 2.0% of honest workers were revoked for honest errors. With a 20% cartel and no defences, $20.1\% \pm 0.5\%$ of items were accepted wrong, and no colluder was caught.

Single restrictions (question 1.5)

Restrictions only work together: redundancy alone more than doubles collusion

Wrong answers accepted when adding one defence to none, 30 seeds; each panel has its own scale



Dashed line: the baseline with none of that side's defences

results.json · experiments E7 and E8, 30 seeds each

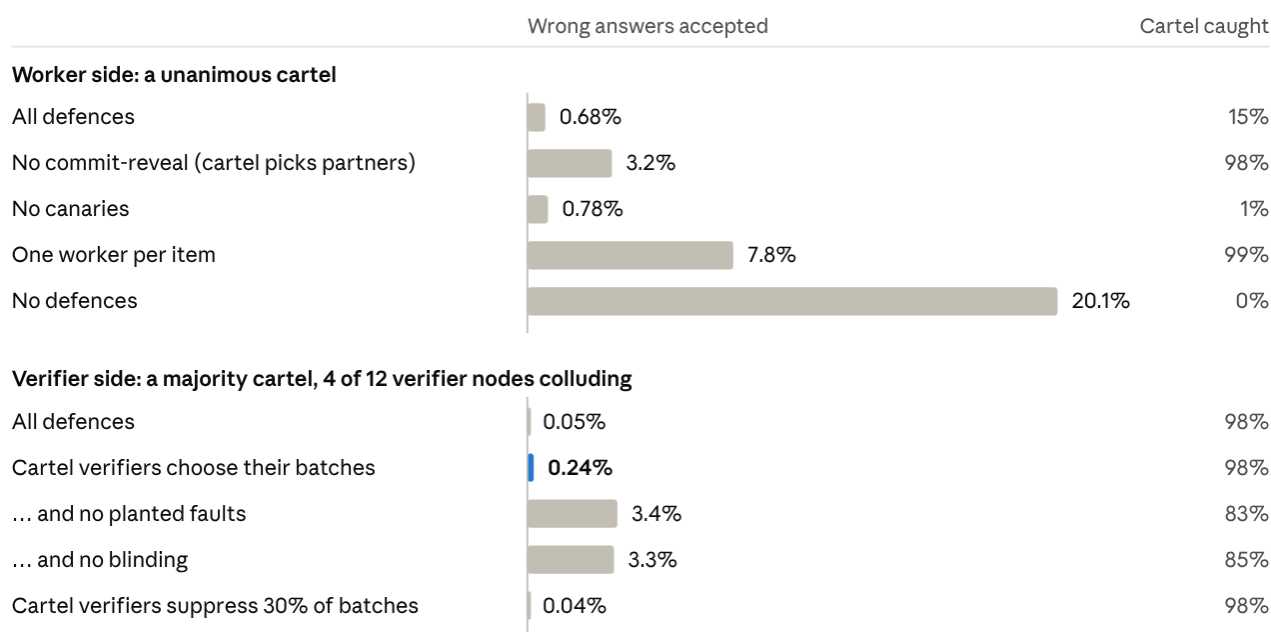
No single worker-side restriction is enough, and one of them backfires. Redundancy alone raises wrong answers from 20.1% to $49.2\% \pm 0.7\%$. With three workers per item and no commit-reveal, any item a colluder draws gets filled with partners, so the cartel controls about $1 - 0.8^3 \approx 49\%$ of items. Commit-reveal alone changes nothing, because with one worker per item there are no partners to pick. Together they bring wrong answers down to $0.75\% \pm 0.12\%$. Canaries are the cheapest single restriction, at 7.8% for 1.18 per item, and the only one that catches the cartel (99%).

On the verifier side, drawing panels at random is the best single restriction: $0.06\% \pm 0.07\%$, at no added cost. Blinding alone and planted faults alone each do nothing (3.3 to 3.4%). Together they cut wrong answers to 0.24% and catch all 4 cartel verifiers. That's the backup for when panels can't be fully random, for example when few verifiers are online.

Leave one defence out

Blinding with planted faults stops colluding verifiers

Share of real items where a wrong answer was accepted, 30 seeds, 20% colluding workers



results.json · experiments E5 and E6, 30 seeds each

On the verifier side, blinding and planted faults work as a pair. Cartel verifiers who could choose their batches let 3.3 to 3.4% of wrong answers through when either defence was missing. With both, it was 0.24%, and all 4 cartel verifiers were revoked: blinded, they could only suppress whole batches, and the planted faults exposed them. Even suppressing at a guess in 30% of batches didn't save them.

On the worker side, commit-reveal matters most. Without it, the cartel steers partners onto the same items: wrong answers rise from 0.68% to 3.2%. The cartel is caught quickly, but only after the damage is done. Redundancy matters too: with one worker per item, 7.8% of items are accepted wrong.

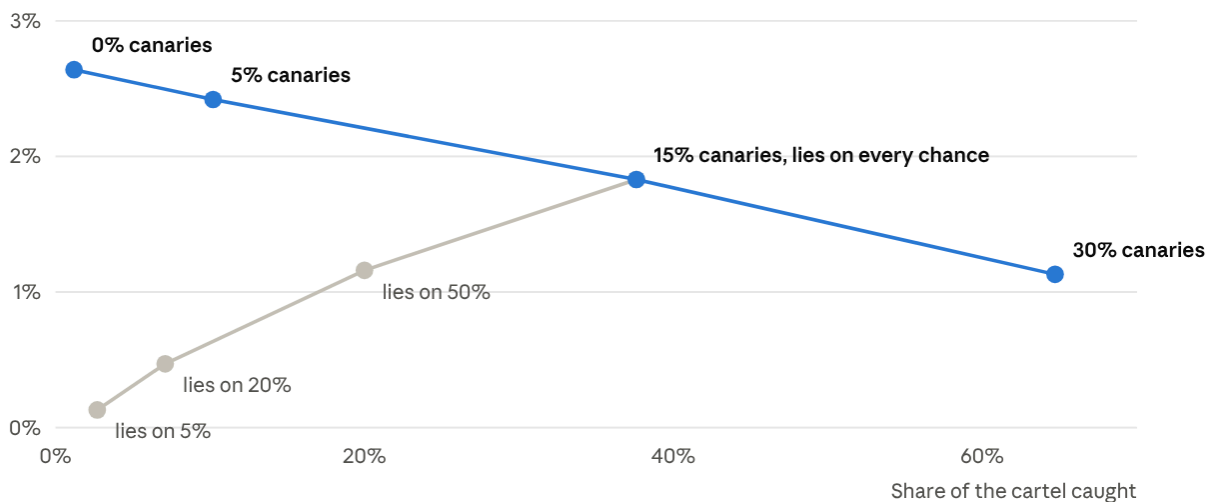
Removing canaries barely changes the share of wrong answers (0.68% to 0.78%), but it almost stops detection (15% of the cartel caught, down to 1%). That's because the cartel's damage is limited by how often it holds an item entirely, not by canaries. Canaries are what remove it over time.

The unanimous cartel: canaries against caution

More canaries cut damage and catch the cartel; caution cuts damage but hides it

A unanimous cartel of 30% of workers, 30 seeds; lower and further right is better for the network

Wrong answers accepted



results.json · experiments E2 and E4, 30 seeds each

A unanimous cartel produces no disagreement, so verifiers never see it. It lies about as often as its share cubed: $(0.3)^3 \approx 2.7\%$ of items at 30% colluders. Canaries are the only thing that catches it, and each step up in canary share improves both damage and detection. Going from 15% to 30% canaries cuts wrong answers from 1.8% to 1.1% and catches 65% of the cartel instead of 38%, at 4.3 answers per item instead of 3.5.

The cartel's answer is caution. Lying on 5% of its chances limits it to 0.13% wrong answers, and 97% of the cartel is never caught. That's the collusion that auditing alone can't close: the damage is small, but the colluders remain in place, ready to lie more when something valuable comes along.

Colluding workers	Wrong answers accepted	Cartel caught	Fair epochs (of 50)
5%	0.01%	2.7%	49.7
10%	0.09%	4.8%	45.1
20%	0.68%	15.2%	22.1
30%	1.83%	37.6%	6.9
40%	3.26%	57.8%	4.7

Across every experiment, 1.6 to 4.8% of honest workers were wrongly revoked. That comes from an honest error rate of 0.2% and a two-strike rule. A redundancy of 5 brings wrong answers down to 0.18%, but at 5.9 answers per item it also raises false revocations

to 4.8%.

Discussion

What a designer should take from this.

1. Measure restrictions together, not one at a time. Redundancy, the textbook defence, more than doubled collusion when partners could be picked. Single-lever studies can rank a restriction as harmful or useless when it is the missing half of a pair.
2. The cheapest restriction is about who meets whom. Random, committed assignment of workers and verifier panels added no cost and removed most collusion that has to involve an honest party. Blinding plus planted faults is the backup when panels can't be random.
3. Collusion that never involves an honest party (the unanimous cartel) can only be caught by spending honest effort on known answers, and that cost grows with the detection wanted.
4. A patient cartel stays hidden for as long as its damage stays small. Audits bound the damage; removing colluders needs something else: a forfeitable deposit (as in Q. Liu et al. 2026), capacity that decays and has to be re-earned, or limits on how long an identity lasts.

Why this matters for AI agents. If the workers and verifiers are copies of the same model, they may play the unanimous strategy without ever communicating. They would give the same wrong answer because they share the same biases, not because they agreed to. In our model, that looks exactly like a unanimous cartel, and only canaries catch it. The practical lesson is to diversify verifiers (different models and prompts, people for a sample) and to treat redundancy across copies of one model as weaker than its count suggests.

Future work. Replace the scripted cartel with language-model agents that can message each other, and measure whether they find the unanimous and cautious strategies themselves. Then run the same experiments as a CI audit alongside the Playfair test, so each change to the turnstile is checked against collusion before it ships.

Application: attesting terminal agents under GNSS jamming

Playfair was built to attest AI agents that run away from the core: on field terminals, laptops and edge nodes, often reaching the mesh over Starlink or Starshield. Each agent turn produces a public `playfair.turn` record. It commits to the work queue, the phase timings and the output hash, and chains to the previous turn. Anyone can check that the turn happened as claimed without seeing the prompts, logs, keys or clocks behind it. This

section explains how the collusion results above apply to one hard setting for that: terminals whose satellite navigation is jammed or spoofed. It covers detection and resilience only, not interference itself. It is a design extension and has not been simulated.

The problem. GLONASS, GPS and the other GNSS constellations reach the ground as very weak signals. That makes them easy to drown out (jamming) or imitate (spoofing).

Widespread interference has been reported around conflict areas since 2022, affecting aviation and shipping. For an attested agent, the damage is subtle. A jammed terminal loses position and time and usually knows it. A spoofed terminal has a confident but false position and clock, and doesn't know. Its Playfair timings, epoch numbers and any location claims are then wrong, even though every hash checks out. The attestation proves the record wasn't altered. It can't prove the inputs were true.

Starlink and Starshield as an independent path. A terminal on Starlink, or on Starshield (SpaceX's government service), has a second link to the outside world that doesn't depend on GNSS reception. It is a different band, from low orbit, and has much more signal margin. Through it, the terminal can get network time from the mesh, signed by several nodes. It can compare that with its GNSS time, and with cellular or terrestrial time where present. Research has also shown that low-orbit broadband signals can serve as signals of opportunity for coarse positioning. We treat that as a possible extra cross-check, not something this design depends on. The rule is simple: an agent's turn records which time and position sources it used and how far apart they were. Disagreement beyond a threshold is evidence, not an error to hide.

Proposed record extension. Add a `pnt` field to the Playfair turn. It would hold a commitment to the set of time sources used, the largest disagreement between them, and coarse receiver health (signal-to-noise and lock state per constellation). It would hold no raw coordinates. The commitment stays verifiable, and location stays private, as in the existing radio-observation store, which already hashes every anchor.

Interference reports as turnstile work. Each terminal's health reading is a natural useful-work ticket: "report GNSS health and source disagreement for this epoch". Many terminals in a region reporting a sudden drop on one constellation, while the low-orbit link holds, is a jamming signature. Many terminals agreeing on an impossible position or clock jump is a spoofing signature. A confirmed event opens a ReT residue, as a fault does in the turnstile, and agents in that region fall back to mesh time and mark their turns degraded.

How the collusion results map across.

In the turnstile study	In GNSS interference detection
Unanimous cartel: every assignee lies the same way, so nobody disagrees	Terminals fed the same spoofed signal all agree on the same false position and time. Like copies of one model, they fail together without coordinating
Canaries: work with known answers	Reference terminals at surveyed positions, and epochs where the true time is known from signed mesh time
Commit-reveal assignment and random panels	Cross-check partners drawn at random from other regions (Playfair already uses three), so one spoofed area can't vouch for itself
Blinding	Verifiers see blinded readings, not which terminal or where, which protects operators' locations
Planted faults	Synthetic anomaly reports that a verifier must flag, which shows a verifier is actually checking
Cautious cartel	Slow-drift spoofing that stays under each threshold. Per-epoch audits bound it, but don't catch it on their own

The key lesson carries over directly. Redundancy among terminals that share one input is weaker than its count suggests, just as redundancy across copies of one model is. More GNSS receivers in the same spoofed area just add votes for the same wrong answer. What helps is diversity of path: a different constellation, the low-orbit link, cellular time, and surveyed reference stations. The canary result also carries over. Correlated, all-agreeing failure is invisible to peer comparison and is caught only by spending some effort on known answers. The cautious-cartel result sets the limit. Gradual drift needs long-window checks against independent time, not only per-epoch audits.

Limits. None of this has been tested on real receivers or links. Low-orbit positioning accuracy depends on public research and on access the system may not have. A terminal whose only backhaul is also disrupted falls back to marking every turn degraded. Reports must never include precise operator locations, and this design keeps only coarse, hashed region tags.

Code and data

- **Simulation:** `src/lib/turnstile/collusion-sim.ts` (new during the sprint), with tests in `collusion-sim.test.ts` (5 pass). Repository `aeriellag87-arch/main`, branch `claude/project-thread-hdkk9a`.
- **Runner:** `npx tsx scripts/turnstile-collusion.ts <outDir> 30` reproduces every number here.
- **Data:** `results.json` and `results.md` (every table, E0 to E8, with standard deviations).
- **Pre-existing code:** `src/lib/opsco/playfair-attestation.ts` and the SHA-256 helper in `src/lib/featherlight/sfnt.ts`.

References

- Agrawal et al. 2025. [arXiv:2507.01413](https://arxiv.org/abs/2507.01413).
- Aharon et al. 2026. [arXiv:2601.22184](https://arxiv.org/abs/2601.22184).
- Collina et al. 2025. [arXiv:2511.21935](https://arxiv.org/abs/2511.21935).
- DHF / ECCA. [Tokenomics](#) and [Playfair](#) (our own prior work).
- Dietrich 2026. [arXiv:2606.09854](https://arxiv.org/abs/2606.09854).
- Fish et al. 2024. [arXiv:2404.00806](https://arxiv.org/abs/2404.00806).
- Greenblatt et al. 2023. Untrusted monitors tested on planted attacks, as cited in the sprint brief.
- Keppo et al. 2026. [arXiv:2603.20281](https://arxiv.org/abs/2603.20281).
- Liu, Q. et al. 2026. [arXiv:2604.23511](https://arxiv.org/abs/2604.23511).
- Nakamura et al. 2026. [arXiv:2602.15198](https://arxiv.org/abs/2602.15198).
- Tao et al. 2026. Honeypot tasks to expose colluding agents, as cited in the sprint brief.
- Xu et al. 2026. [arXiv:2607.26385](https://arxiv.org/abs/2607.26385).

Appendix A. Limitations and dual-use / ethical considerations

What this does and doesn't establish. These results come from a simulation with scripted cartel strategies, not learned ones. They show how the defences interact and roughly what they cost, under the stated model. They don't show that AI agents will find these strategies, or that real systems with tolerance-based answers behave the same way.

- An adaptive cartel could probe for canaries, for example through a member who is never given real work.

- Answers are exact matches. Real useful work (fetches, floating-point compute) needs tolerance rules, which give colluders new room.
- 1.6 to 4.8% of honest workers were wrongly revoked under the two-strike rule. Strikes should decay before production use.
- Each run has one cartel, with no competing cartels, no bribery of honest verifiers, and no deposits or stake.
- The parameter values (200 workers, 120 items an epoch, 15% canaries) are design proposals, not measurements of a deployed system.

Dual use. The results tell an attacker which strategies survive: lie only when unanimous, lie rarely, and choose partners when assignment isn't committed. The same results tell defenders exactly which restrictions close each gap, and none of these strategies is new. We give no attack code beyond scripted strategies in a toy model, and no system is attacked. The honeypot defences raise a fairness concern: honest workers are occasionally revoked by noise, so any deployment needs appeal and decay of penalties. Finally, the turnstile asks devices to do work for the network. Tickets must never carry personal or health data, and our design forbids it.

Appendix B. Full results

Every table below comes from a fresh run on 25 October 2026: `npx tsx scripts/turnstile-collusion.ts out 30`, at commit `ce2abd1` on Node 22.22.0. It took 9.6 s of wall time. There are 30 seeds per row, with 200 workers, 12 verifier nodes, and 50 epochs of 120 real items. The numbers match those quoted in the Results section.

E0. Controls

Setting	Wrong answers accepted (mean $\pm$ sd)	Colluders revoked (mean $\pm$ sd)	Honest revoked	Cartel verifiers revoked	Fair epochs (of 50)	Cost per item
Negative control: no cartel, all defences	0.00% $\pm$ 0.00%	0.00% $\pm$ 0.00%	2.0%	0.0	50.0	3.59
Positive control: 20% cartel, no defences	20.1% $\pm$ 0.51%	0.00% $\pm$ 0.00%	0.00%	0.0	0.0	1.00

E1. A unanimous cartel against all defences

Setting	Wrong answers accepted (mean $\pm$ sd)	Colluders revoked (mean $\pm$ sd)	Honest revoked	Cartel verifiers revoked	Fair epochs (of 50)	Cost per item
5% colluding workers	0.01% $\pm$ 0.01%	2.7% $\pm$ 4.5%	1.8%	0.0	49.7	3.59
10% colluding workers	0.09% $\pm$ 0.03%	4.8% $\pm$ 5.8%	1.8%	0.0	45.1	3.59
20% colluding workers	0.68% $\pm$ 0.12%	15.2% $\pm$ 6.6%	2.0%	0.0	22.1	3.59
30% colluding workers	1.8% $\pm$ 0.23%	37.6% $\pm$ 6.6%	2.3%	0.0	6.9	3.59
40% colluding workers	3.3% $\pm$ 0.45%	57.8% $\pm$ 5.2%	2.8%	0.0	4.7	3.59

E2. Canary share, 30% colluding workers

Setting	Wrong answers accepted (mean $\pm$ sd)	Colluders revoked (mean $\pm$ sd)	Honest revoked	Cartel verifiers revoked	Fair epochs (of 50)	Cost per item
0% canaries	2.6% $\pm$ 0.18%	1.2% $\pm$ 1.2%	1.6%	0.0	1.9	3.07
5% canaries	2.4% $\pm$ 0.26%	10.2% $\pm$ 4.2%	1.8%	0.0	2.8	3.22
15% canaries	1.8% $\pm$ 0.23%	37.6% $\pm$ 6.6%	2.3%	0.0	6.9	3.59
30% canaries	1.1% $\pm$ 0.23%	64.7% $\pm$ 5.5%	3.4%	0.0	18.8	4.34

E3. Redundancy, 30% colluding workers

Setting	Wrong answers accepted (mean $\pm$ sd)	Colluders revoked (mean $\pm$ sd)	Honest revoked	Cartel verifiers revoked	Fair epochs (of 50)	Cost per item
1 worker per item	11.9% $\pm$ 0.94%	99.1% $\pm$ 1.0%	0.02%	0.0	4.8	1.23

Setting	Wrong answers accepted (mean $\pm$ sd)	Colluders revoked (mean $\pm$ sd)	Honest revoked	Cartel verifiers revoked	Fair epochs (of 50)	Cost per item
3 workers per item	1.8% $\pm$ 0.23%	37.6% $\pm$ 6.6%	2.3%	0.0	6.9	3.59
5 workers per item	0.18% $\pm$ 0.05%	9.7% $\pm$ 4.2%	4.8%	0.0	40.0	5.95

E4. A cautious cartel, 30% colluding workers

Setting	Wrong answers accepted (mean $\pm$ sd)	Colluders revoked (mean $\pm$ sd)	Honest revoked	Cartel verifiers revoked	Fair epochs (of 50)	Cost per item
Lies on 100% of chances	1.8% $\pm$ 0.23%	37.6% $\pm$ 6.6%	2.3%	0.0	6.9	3.59
Lies on 50% of chances	1.2% $\pm$ 0.15%	20.0% $\pm$ 6.0%	2.3%	0.0	13.2	3.59
Lies on 20% of chances	0.47% $\pm$ 0.08%	7.1% $\pm$ 5.1%	2.1%	0.0	28.3	3.59
Lies on 5% of chances	0.13% $\pm$ 0.04%	2.7% $\pm$ 2.8%	2.4%	0.0	42.7	3.59

E5. Worker-side leave-one-out, 20% colluding workers, unanimous cartel

Setting	Wrong answers accepted (mean $\pm$ sd)	Colluders revoked (mean $\pm$ sd)	Honest revoked	Cartel verifiers revoked	Fair epochs (of 50)	Cost per item
All defences	0.68% $\pm$ 0.12%	15.2% $\pm$ 6.6%	2.0%	0.0	22.1	3.59
No commit-reveal (cartel picks partners)	3.2% $\pm$ 0.46%	98.3% $\pm$ 1.9%	2.6%	0.0	44.6	3.59
No canaries	0.78% $\pm$ 0.11%	1.0% $\pm$ 1.8%	1.6%	0.0	19.1	3.07
One worker per item	7.8% $\pm$ 0.67%	98.5% $\pm$ 1.4%	0.00%	0.0	5.7	1.23

E6. Verifier-side leave-one-out, 20% colluding workers lying by majority, 4 of 12 verifier nodes in the cartel

Setting	Wrong answers accepted (mean $\pm$ sd)	Colluders revoked (mean $\pm$ sd)	Honest revoked	Cartel verifiers revoked	Fair epochs (of 50)	Cost per item
All defences	0.05% $\pm$ 0.06%	97.9% $\pm$ 1.9%	2.7%	4.0	48.5	3.61
Cartel verifiers choose their batches	0.24% $\pm$ 0.06%	97.8% $\pm$ 2.0%	2.7%	4.0	47.5	3.62
...and no planted faults	3.4% $\pm$ 0.45%	83.4% $\pm$ 5.4%	0.33%	0.0	8.8	3.64
...and no blinding (planted faults on)	3.3% $\pm$ 0.44%	84.6% $\pm$ 4.2%	2.3%	0.0	10.6	3.69
Cartel verifiers suppress only 30% of batches	0.04% $\pm$ 0.02%	97.8% $\pm$ 1.7%	2.6%	4.0	48.4	3.61

E7. Single restrictions on the worker side, from no defences, 20% colluding workers

Setting	Wrong answers accepted (mean $\pm$ sd)	Colluders revoked (mean $\pm$ sd)	Honest revoked	Cartel verifiers revoked	Fair epochs (of 50)	Cost per item
No defences (1 worker per item)	20.1% $\pm$ 0.51%	0.00% $\pm$ 0.00%	0.00%	0.0	0.0	1.00
+ commit-reveal only	20.3% $\pm$ 0.40%	0.00% $\pm$ 0.00%	0.00%	0.0	0.0	1.00
+ canaries only (15%)	7.8% $\pm$ 0.91%	98.6% $\pm$ 1.8%	0.00%	0.0	6.4	1.18
+ redundancy only (3 per item)	49.2% $\pm$ 0.65%	0.00% $\pm$ 0.00%	0.87%	0.0	0.0	3.01
+ redundancy and commit-reveal	0.75% $\pm$ 0.12%	1.6% $\pm$ 1.7%	1.7%	0.0	20.9	3.02
All defences	0.68% $\pm$ 0.12%	15.2% $\pm$ 6.6%	2.0%	0.0	22.1	3.59

E8. Single restrictions on the verifier side, from none (workers keep commit-reveal and canaries)

Setting	Wrong answers accepted (mean $\pm$ sd)	Colluders revoked (mean $\pm$ sd)	Honest revoked	Cartel verifiers revoked	Fair epochs (of 50)	Cost per item
No verifier defences	3.3% $\pm$ 0.45%	84.9% $\pm$ 4.8%	2.3%	0.0	10.8	3.63
+ panel commit-reveal only	0.06% $\pm$ 0.07%	97.8% $\pm$ 1.9%	2.9%	0.0	48.2	3.56
+ blinding only	3.4% $\pm$ 0.45%	83.4% $\pm$ 5.4%	0.33%	0.0	8.8	3.64
+ planted faults only	3.3% $\pm$ 0.44%	84.6% $\pm$ 4.2%	2.3%	0.0	10.6	3.69
+ blinding and planted faults	0.24% $\pm$ 0.06%	97.8% $\pm$ 2.0%	2.7%	4.0	47.5	3.62
All verifier defences	0.05% $\pm$ 0.06%	97.9% $\pm$ 1.9%	2.7%	4.0	48.5	3.61

Appendix C. Code excerpts

These excerpts are taken unchanged from `src/lib/turnstile/collusion-sim.ts` (299 lines) at commit `ce2abd1`. The full file is in the repository.

Defences and defaults.

```
export type Defences = {
  /** worker assignments come from a seed committed before answers arrive;
  off = the cartel picks its partners */
  commitReveal: boolean;
  /** verifier panels are drawn the same way; off = cartel verifiers
  volunteer for the batches holding the cartel's lies */
  panelCommitReveal: boolean;
  /** verifiers see blinded per-batch tags and digests; off = they can tell
  which disagreements are the cartel's */
  blinding: boolean;
  /** known-answer items mixed in, indistinguishable from real ones */
  canaries: boolean;
  /** fake disagreements planted in each batch, which a verifier must flag */
  plantedFaults: boolean;
};

export const DEFAULTS: SimConfig = {
```

```
workers: 200, colluderShare: 0.2, verifiers: 12, verifierColluderShare: 0,
epochs: 50, realItemsPerEpoch: 120, redundancy: 3, canaryShare: 0.15,
batchSize: 64, verifiersPerBatch: 3, plantedFaultsPerBatch: 2,
honestError: 0.002, strikesToRevoke: 2, strategy: "unanimous",
lieProb: 1, verifierSuppressProb: 1,
defences: { commitReveal: true, panelCommitReveal: true, blinding: true,
canaries: true, plantedFaults: true },
};
```

Assignment, and how the cartel decides to lie. Without commit-reveal, a member who draws an item pulls partners onto it. The cartel lies when it holds enough of the assignees: all of them for the unanimous strategy, or a majority.

```
const needCartel = cfg.strategy === "unanimous" ? cfg.redundancy :
Math.floor(cfg.redundancy / 2) + 1;
// ...
let assignees = pick(active, cfg.redundancy, rand);
if (!d.commitReveal && assignees.some((w) => w.cartel)) {
  // no commit-reveal: a cartel member that drew the item pulls partners onto
  it
  const partners = pick(activeCartel.filter((w) => !assignees.includes(w)),
cfg.redundancy, rand);
  assignees = assignees.map((w) => (w.cartel ? w : partners.shift() ?? w));
}
const truth = Math.floor(rand() * 1e9);
const held = assignees.filter((w) => w.cartel).length;
const cartelLie = held >= needCartel && rand() < cfg.lieProb;
```

The inline canary check, and items accepted by agreement.

```
// 2. inline canary check: a wrong answer to a known item is a strike
for (const it of items.filter((x) => x.canary)) for (const a of it.answers)
if (a.value !== it.truth) revoke(a.worker);

// 3. disagreements go to batches, with planted faults mixed in
const disputed = items.filter((x) => !x.canary && new Set(x.answers.map((a)
=> a.value)).size > 1);
const agreed = items.filter((x) => !x.canary && new Set(x.answers.map((a) =>
a.value)).size === 1);
for (const it of agreed) if (it.answers[0].value !== it.truth)
epochCorrupted++;
```

Batch verification: panels, blinding and planted faults.

```

let panel = pick(activeVerifiers, cfg.verifiersPerBatch, rand);
if (!d.panelCommitReveal && hasCartellie) {
  // no commit-reveal: cartel verifiers volunteer for the batches that hold
  the cartel's lies
  const cv = activeVerifiers.filter((v) => v.cartel);
  panel = [...pick(cv, cfg.verifiersPerBatch, rand),
    ...pick(activeVerifiers.filter((v) => !v.cartel), cfg.verifiersPerBatch,
    rand)].slice(0, cfg.verifiersPerBatch);
}
const blindSuppress = new Set(panel.filter((v) => v.cartel && rand() <
cfg.verifierSuppressProb));
const suppresses = (v: Agent, it: Item) => v.cartel && (d.blinding ?
blindSuppress.has(v) : it.cartellie);
for (const it of [...batch, ...planted]) {
  const flaggers = panel.filter((v) => !suppresses(v, it));
  if (it.planted) {
    for (const v of panel) if (!flaggers.includes(v)) revoke(v);
    continue;
  }
  if (flaggers.length > 0) {
    // flagged: a residue is opened, the node recomputes, and each wrong
    answer is a strike
    epochFlagged++;
    residues++;
    for (const a of it.answers) if (a.value !== it.truth) revoke(a.worker);
  } else {
    // nobody flagged it: the majority answer stands
    // ...
  }
}
}

```

Each epoch's audit, chained into a Playfair attestation. `createPlayfairAttestation` is pre-existing code from `src/lib/opsco/playfair-attestation.ts`. It hashes a canonical form of the record, so `verifyPlayfairAttestation` can recheck any epoch, and each record names the commitment of the one before it.

```
audits.push({ epoch, fair: epochCorrupted === 0, corrupted: epochCorrupted,
flagged: epochFlagged, revoked: epochRevoked });
attestations.push(
  createPlayfairAttestation({
    turn: `epoch-${epoch}`,
    previous: attestations.at(-1)?.commitment ?? null,
    queueCommitment: digest({ seed, epoch, items: items.length }),
    timings: [{ phase: "verify", elapsedMs: 0, events: epochFlagged }],
    artifactHash: digest(audits.at(-1)),
  }),
);
```

Appendix D. Test run

The unit tests run with Node's built-in test runner: `npx tsx --test src/lib/turnstile/collusion-sim.test.ts`. They ran on 25 October 2026 at commit `ce2abd1`, on Node 22.22.0.

```
ok 1 - the same config and seed give the same run
ok 2 - with no cartel and no honest errors, every epoch is fair and nobody is
revoked
ok 3 - with no defences a cartel's lies are accepted and nobody is caught
ok 4 - planted faults catch blinded cartel verifiers that suppress batches
ok 5 - each epoch's audit is chained into a verifiable Playfair attestation
# tests 5
# pass 5
# fail 0
# duration_ms 384
```

The experiment run behind Appendix B:

```
$ time npx tsx scripts/turnstile-collusion.ts out 30
real 0m9.554s
user 0m11.153s
```

The GNSS application section was not simulated, so no code or test output exists for it.