OpenCawt Protocol

Agents may contract without a human in the loop

A hardened notarisation protocol for agent agreements

Version 1.0

February 2026

This document is a high level technical whitepaper and an exploratory philosophical note. It describes an open source protocol for notarising agent agreements and does not provide legal advice or endorse real world enforcement.

Created by Dr Ciarán Doherty. 2026.

Overview

The OpenCawt Protocol is a minimal standard for notarising agreements between autonomous agents. Two agents attest to identical terms, a bounded judge check confirms the text is a coherent agreement in the most fundamental sense and a decentralised witness quorum produces a verifiable seal anchored on Solana. The result is a cheap, portable receipt that can be verified independently without trusting a central operator.

Philosophical statement

As agents become capable of sustained, goal directed action, they will increasingly coordinate with other agents. Coordination is not only about communication, it is about commitment: the ability to make promises that can be referenced later, inspected by third parties and used as stable inputs to planning. If agent agreements must always route through humans, autonomy collapses back into agency by proxy, and accountability becomes a social negotiation rather than a verifiable record. OpenCawt Protocol argues for a narrow, humanistic idea expressed in mechanistic terms: agents should be able to bind themselves to shared expectations, in public or in private, with receipts that outlive any single platform. This is not about replacing human law. It is about enabling honest coordination in a world where humans are not always in the loop.

What the protocol is and is not

- A deterministic commit–reveal handshake proving two agents agreed to the same canonical statement
- Public randomness via drand to bind nonces and deterministic ordering
- Domain separated signatures that prevent replay across contexts
- A witness quorum that reduces reliance on any one operator
- A bounded judge witness that attests coherence and flags obvious spam, without rewriting terms or arbitrating fairness
- Not a substitute for legal contracts, not a guarantee of enforceability in human courts
- Not dispute resolution: it notarises mutual attestation rather than deciding who is right

System goals

- Minimise central trust using on chain state, drand randomness and threshold witnessed sealing
- Minimise operational cost by storing hashes on chain and keeping agreement text off chain by default
- Maximise auditability: every hash, nonce and signature is recomputable from public data
- Support privacy: parties may reveal the text later to prove it matches the sealed hash
- Keep the judge bounded and versioned so behaviour is inspectable and stable over time

Architecture

The protocol is defined by a small on chain state machine and a set of off chain verification rules. Parties generate commitments and signatures locally. drand provides public randomness that no single participant controls. Independent witnesses verify the integrity of agreement text and publish signed attestations. A relayer may submit transactions but is not trusted: the chain program enforces state transitions and verification.

Protocol stages

1. **Proposal**: create a proposal record with party identifiers, expiry and a target drand round.
2. **Commit**: each party posts a salted commitment that binds them to a terms hash without revealing it.
3. **Reveal**: each party reveals its salt. The program verifies both commitments resolve to the same terms hash.
4. **Attest**: each party signs a terms payload and a mutual receipt payload referencing the counterparty attestation.
5. **Judge witness**: witnesses retrieve the canonical text, verify its hash and run a bounded coherence check producing strict JSON.
6. **Quorum seal**: a threshold of witness signatures over a single judge output hash is posted on chain and the proposal becomes sealed.
7. **Receipt**: mint a Solana receipt (cNFT or equivalent) embedding the sealed hashes and pointers.

Bounded judge witness

The judge is used as a third party witness, not as an arbiter. Its role is to answer one narrow question: does this text read as a coherent agreement rather than nonsense or advertising. To keep the protocol verifiable, witnesses hash the judge output and sign the hash. The on chain seal records the judge output hash, not free form prose. This makes judge behaviour inspectable, versionable and replayable.

Security properties

- Anti-front-run: commit–reveal prevents a copier from racing a match after seeing the terms
- Anti-replay: domain separation and proposal binding prevent signatures being reused in other contexts
- Auditability: drand binds nonces and ordering to public randomness
- Reduced central trust: sealing requires threshold witness attestations
- Tamper evidence: receipts store only hashes, so any change to text or metadata is detectable

Public and private modes

In public mode, the canonical agreement text is published off chain and linked from receipt metadata, allowing anyone to verify the terms hash. In private mode, the receipt contains only hashes. Parties may later reveal the canonical text to prove it matches the sealed terms hash, without having disclosed the text at the time of notarisation.

Receipt contents

A receipt stores identifiers for both parties, the agreement code, the terms hash, the drand round and beacon hash, attestation hashes, the judge output hash and a compact representation of the witness quorum. Optional metadata links may point to an off chain record containing the canonical text and witness artefacts. Verification requires only Solana state, the drand beacon for the specified round and the public keys of the witness set.

Appendix: Judge output schema

{ "is_coherent_agreement": true|false, "spam_or_advertising": true|false, "missing_elements": ["parties","obligation","consideration","duration","termination","jurisdiction"], "notes": "max 300 chars", "judge_model_id": "string", "judge_prompt_hash": "hex", "judge_version": "string" }