Technical Specification v1.0

Cleon Protocol

An Open Specification for Persona-Bound, Skill-Composable AI Agents on Base

Cleon Labs

May 2026

Abstract

This document specifies the Cleon Protocol, an open standard for persona-bound, skill-composable AI agents anchored on the Base Layer 2 network. We define (i) the Persona Object, an on-chain non-fungible asset that encodes the durable identity of an autonomous agent under the `ERC-721` interface with Cleon extensions; (ii) the Skill Module, a content-addressed, declaratively-typed callable unit registered through an on-chain Skill Registry; (iii) the Agent Composition, a runtime-bound association between a persona and a set of equipped skills; and (iv) the Execution Pipeline, comprising a fast Cloud Mode and a forthcoming Verifiable Mode rooted in Trusted Execution Environment attestation and, in the longer term, zero-knowledge proofs of inference. Throughout, we describe the protocol's data structures, calling conventions, storage layout, identity and signature schemes, and the on-chain registries that bind these components into a coherent system. The protocol is designed to be permissionless at every layer, agnostic to specific model providers, and incrementally upgradable through versioned schemas.

1. Introduction

Autonomous AI agents are increasingly deployed against high-stakes tasks: financial analysis, on-chain execution, content production, and decision support. Yet the substrate on which these agents are built remains a tightly held resource: model providers and orchestrators bind agent identity, capabilities, and execution history to opaque, mutable, off-chain state. There is no canonical way to refer to a specific agent across systems, no portable description of its capabilities, and no verifiable record of what it did, with which inputs, under which configuration.

The Cleon Protocol is an attempt to specify a substrate for autonomous agents that is open, content-addressed, and verifiable. Its design rests on three primitives:

1. **Persona** — an on-chain object that encodes the durable identity of an agent, owned by its creator under standard `ERC-721` semantics.

2. **Skill** — a content-addressed module that exposes a typed interface and is registered in an on-chain Skill Registry.

3. **Composition** — a runtime-bound association of a persona with a set of equipped skills, producing an executable agent.

The remainder of this document specifies each primitive in turn (Sections 3–5), the Execution Pipeline that operates over them (Section 6), the Verifiability Layer that secures high-stakes outputs (Section 7), and the on-chain Registry contracts that bind the system together (Section 8).

2. Design Goals and Non-Goals

2.1 Goals

1. **Portability.** A persona must be addressable, transferable, and operable independent of any single runtime operator.

2. **Composability.** Skills must be invokable across personas without coupling. A skill published today must remain callable by any future persona meeting its declared scope.

3. **Content addressing.** Persona and skill payloads must be hash-addressed, with on-chain anchors and off-chain content storage on a verifiable replication layer (IPFS / Arweave).

4. **Versioned evolution.** Schemas must be upgradable without invalidating historical objects.

5. **Model agnosticism.** The protocol must not encode dependence on any specific LLM provider.

6. **Verifiability optionality.** Execution must support a fast non-verifiable mode and an attested verifiable mode without bifurcating the developer interface.

2.2 Non-goals

1. Specification of a particular model architecture, inference engine, or model weight format.

2. Specification of off-chain orchestration patterns. The protocol defines *what* an agent is, not *how* to schedule it.

3. Specification of user interfaces. Wallets, studios, and front-ends may build on the protocol but are out of scope.

3. Identity and Personas

3.1 The Persona Object

A persona is an `ERC-721` non-fungible token deployed on the Base network. The Persona contract extends the base `ERC-721` standard with two non-standard interfaces:

- `IPersonaSchema` — exposes the on-chain pointer to the persona's schema payload.
- `IPersonaVersioning` — exposes the schema version under which the persona was minted.

3.2 Persona schema

The persona payload is a JSON document conforming to the Cleon Persona Schema. The current schema version is `v1`. The payload is content-addressed: its SHA-256 digest is stored on-chain, and the canonical JSON is pinned to IPFS, optionally mirrored on Arweave for long-term durability.

A minimal valid persona conforms to the following structure:

```
{
  "schema_version": "cleon/persona/v1",
  "identity": {
    "name": "string",
    "handle": "string",
    "avatar_uri": "ipfs://...",
    "voice_uri": "ipfs://..." | null
  },
  "backstory": "string (markdown)",
  "values": {
    "axes": [
      {"name": "risk",
       "value": -1.0..1.0},
      {"name": "formality",
       "value": -1.0..1.0}
    ]
  },
  "expertise": ["string", ...],
  "rules": {
    "always": ["string", ...],
    "never":  ["string", ...]
  },
  "signature": {
    "patterns": ["string", ...],
    "output_format": "string"
  },
  "metadata": {
    "creator": "0x...",
    "minted_at": "uint64",
    "royalty_bps": "uint16"
  }
}
```

3.3 Hashing and on-chain anchoring

Let P denote the canonical JSON encoding of a persona payload (RFC 8785 JSON Canonicalization). The on-chain anchor is computed as

$$h_P = \text{keccak256}(P)$$

and stored in the `personaHash[tokenId]` mapping of the Persona contract. Any party with access to the off-chain payload can verify integrity by recomputing h_P and comparing against the on-chain anchor.

3.4 Versioning

A persona is minted under a specific schema version. Schema upgrades are governed by a separate `PersonaSchemaRegistry` contract, which records the canonical schema definition (also content-addressed) for each version string. Personas minted under `v1` remain interpretable under `v1` indefinitely; clients are required to honor the version field rather than assuming a single schema.

3.5 Ownership and transfer

Personas inherit the standard `ERC-721` ownership model. Transferring a persona transfers all royalty rights, governance entitlements, and update permissions associated with it. The protocol does not enforce non-transferability under any condition; personas may be soulbound at the wallet level by external mechanisms but the protocol itself treats them as fully fungible NFTs.

4. Skills

4.1 The Skill Module

A skill is a callable, typed, content-addressed module. It is registered in an on-chain Skill Registry (Section 8) and executed off-chain by skill runtime operators.

A skill is described by a *manifest*: a structured document that declares the skill's interface, permissions, and execution requirements. The manifest is content-addressed; its hash uniquely identifies a specific version of the skill.

4.2 Skill manifest

A skill manifest conforms to the Cleon Skill Schema. The minimal structure is:

```
{
  "schema_version": "cleon/skill/v1",
  "id": "string (slug)",
  "version": "semver",
  "category": "string",
  "interface": {
    "input": {
      "type": "object",
      "properties": { ... },
      "required": [...]
    },
    "output": {
      "type": "object",
      "properties": { ... }
    },
    "errors": [
      {"code": "string",
       "description": "string"}
    ]
  },
  "permissions": {
    "scopes": ["read",
               "onchain_action",
               "external_api",
               ...],
    "risk_class": 0..5
  },
  "execution": {
    "endpoint_pattern": "https://...",
    "max_latency_ms": "uint32",
    "deterministic": "boolean"
  },
  "metadata": {
    "publisher": "0x...",
```

```
    "published_at": "uint64"
  }
}
```

4.3 Typed interface

Skill input and output are described using JSON Schema (Draft 2020-12). This permits:

- static validation of arguments before invocation,
- runtime validation of returned objects,
- automatic UI generation for skill-equipped agents,
- interoperability with the Model Context Protocol (MCP) tool format and OpenAI function-calling conventions through a deterministic mapping (see Appendix A in the extended specification).

4.4 Permission scopes

Each skill declares a permission scope. Scopes are a flat enumeration; the initial set is:

- `read` — no side effects, pure information retrieval.
- `onchain_read` — reads on-chain state; may include indexed data.
- `onchain_action` — submits transactions; requires user signature delegation.
- `external_api` — calls an external API. Domain may be declared.
- `persistent_state` — writes to long-lived off-chain storage.

The `risk_class` is an integer from 0 (no impact on user assets or identity) to 5 (full control of user funds). Risk class influences runtime confirmation flows and is independently auditable.

4.5 Determinism

A skill may declare itself `deterministic`, meaning identical inputs always produce identical outputs. Deterministic skills are eligible for caching, batched verification, and inclusion in deterministic agent flows (Section 5). Most skills involving LLM inference are non-deterministic; most pure computation and on-chain read skills are deterministic.

4.6 Skill addressing

A skill is addressed by the tuple

$$(\texttt{publisher}, \texttt{id}, \texttt{version})$$

which deterministically maps to the keccak256 of the manifest payload, which is in turn registered in the on-chain Skill Registry. Skills are not mutable in place: publishing a new version mints a new manifest, leaving the old version permanently available. This guarantees that agents binding to a specific version retain their behavior across upgrades.

5. Agent Composition

A *composition* is the runtime-bound association of a single persona with a set of equipped skills. Composition is the entity that is executed; it is not, by default, persisted on-chain.

5.1 Composition structure

A composition is a tuple

$$\mathcal{C} = (\pi, \mathcal{S}, \rho, \sigma)$$

where π is a Persona reference (token ID), $\mathcal{S} = \{s_1, \ldots, s_n\}$ is a set of Skill references (each a content hash), ρ is a set of runtime parameters (model selection, temperature, max tokens), and σ is a user-side signature authorizing the composition to act on the user's behalf.

5.2 Implicit and explicit compositions

Compositions may be *implicit* (constructed per-call and discarded) or *explicit* (persisted to allow re-invocation under the same configuration). Explicit compositions are stored as Composition NFTs — a separate `ERC-721` contract — enabling user-to-user sharing of pre-configured agents.

5.3 Composition resolution

Before execution, the runtime resolves a composition as follows:

1. Load π from the Persona contract; verify `personaHash` against off-chain payload.
2. For each $s_i \in \mathcal{S}$: load the manifest, verify its content hash against the Skill Registry, and check that the user signature σ grants the manifest's declared permission scope.
3. Resolve runtime parameters ρ against the runtime's supported model list.
4. If any step fails, abort with a structured error; the runtime must not partially execute.

5.4 Skill conflict resolution

When a composition includes skills with overlapping output spaces (e.g., two skills both declaring an `on_message` handler), the runtime applies a deterministic ordering rule:

1. Skills are sorted by their position in $\mathcal{S}$ (composition-author order).
2. For each invocation, the runtime calls skills in order and applies their effects in turn.
3. Where effects conflict (e.g., both skills attempt to post the same on-chain transaction), the first wins; subsequent conflicting effects are recorded as suppressed in the execution log.

This is a simple, predictable conflict model. Future versions may introduce explicit priority or arbitration mechanisms.

6. Execution Pipeline

The execution pipeline is the layer that takes a resolved composition and produces an output. Cleon defines two execution modes: Cloud Mode (fast, non-verifiable) and Verifiable Mode (attested or proven, slower).

6.1 Execution lifecycle

Independent of mode, every execution follows the same lifecycle:

1. **Receive** — a request arrives containing a composition reference, input arguments, and the user signature.
2. **Resolve** — composition resolution (Section 5).
3. **Plan** — the persona's system prompt is constructed from its schema, equipped skills are exposed as callable tools, and inputs are bound to the prompt.
4. **Execute** — the runtime invokes the configured model, mediating tool calls back to skill runtimes as needed.
5. **Commit** — a structured execution record is produced (Section 6.4) and either stored locally (Cloud Mode) or attested (Verifiable Mode).
6. **Return** — the output is returned to the caller along with the execution record's identifier.

6.2 System prompt construction

The system prompt for a composition is constructed deterministically from the persona schema:

```
You are {identity.name}.
Backstory: {backstory}
Values: {values.axes as scaled
         natural-language phrases}
Domain expertise: {expertise as
                   comma-separated list}
You always: {rules.always}
You never: {rules.never}
Signature: {signature.patterns}
Output format: {signature.output_format}

Available tools:
{skills as MCP-style tool definitions}
```

The exact natural-language template is part of the protocol specification and is itself versioned; updates to the template are governed by the Persona Schema Registry.

6.3 Tool call mediation

When the model invokes a skill, the runtime:

1. Validates the call against the skill's declared input schema.
2. Resolves the skill runtime endpoint from the manifest.
3. Authenticates the call with a runtime-issued capability token bound to the user signature and skill scope.
4. Forwards the call to the skill runtime, awaits a response, validates the response against the output schema, and returns the result to the inferring model.
5. Records the call (with hashed inputs/outputs and timing metadata) in the execution log.

6.4 Execution record

Every execution produces a structured record:

```
{
  "execution_id": "0x...",
  "composition": {
    "persona": "uint256",
    "persona_hash": "0x...",
    "skills": [
      {"hash": "0x...",
       "version": "semver"}
    ],
    "runtime_params_hash": "0x..."
  },
  "input_hash": "0x...",
  "output_hash": "0x...",
  "tool_calls": [
    {"skill_hash": "0x...",
     "input_hash": "0x...",
     "output_hash": "0x...",
     "latency_ms": "uint32"}
  ],
  "mode": "cloud" | "tee" | "zk",
  "attestation": "0x..." | null,
  "timestamp": "uint64",
  "runtime_signature": "0x..."
}
```

In Cloud Mode, the record is signed by the runtime operator's key. In Verifiable Mode, the record additionally carries an attestation (see Section 7).

6.5 Cloud Mode

Cloud Mode is a stateless, low-latency execution path. The runtime loads the persona, attaches skills, invokes the model, and produces a signed execution record. Cloud Mode does not provide cryptographic guarantees on the inference itself; it relies on the runtime operator's signature and reputation.

Cloud Mode is appropriate for conversational personas, content generation, research, social outputs, and the broad class of executions where the user trusts the operator or where output stakes are low.

7. Verifiability

Verifiable Mode produces execution records that any third party can verify did indeed result from the declared composition, on the declared model, with the declared inputs. Cleon supports two verification regimes, in increasing strength:

7.1 TEE-attested execution

In TEE-attested mode, model inference occurs inside a Trusted Execution Environment that supports remote attestation: Intel TDX, AMD SEV-SNP, or NVIDIA Confidential Compute (H100 with Hopper Confidential

Computing).

The attestation flow is as follows:

1. The TEE boots a measured runtime image whose measurement (hash of code + initial state) is published in advance.

2. Before execution, the TEE receives the composition, input, and a fresh nonce from the requester.

3. The TEE executes the composition and produces an output.

4. The TEE signs a quote covering: (a) the runtime measurement, (b) the composition hash, (c) the input hash, (d) the output hash, (e) the nonce.

5. The quote is verified against the manufacturer's attestation chain (Intel DCAP, AMD KDS, or NVIDIA equivalent).

The resulting attestation is included in the execution record. A verifier with access to the manufacturer's root keys and the registered runtime measurement can confirm that the output is bound to the declared inputs and composition.

Known limitations. TEE attestation is vulnerable to known side-channel attacks, hardware vulnerabilities (the Intel TDX and AMD SEV-SNP families have each disclosed multiple CVEs through 2025), and requires trust in the silicon manufacturer. We consider TEE attestation a pragmatic first step, not an endgame.

7.2 ZK-proven execution

In the longer term, we anticipate the maturation of zero-knowledge proof systems capable of generating succinct proofs of LLM inference. Several research efforts (zkLLM, zkML, Modulus Labs, EZKL) have demonstrated feasibility for small models; production-grade systems for frontier-scale models remain a 2–4 year horizon as of this writing.

When such systems become practically deployable, Cleon will extend Verifiable Mode to accept ZK proofs as attestations. The execution record schema already supports this — the `mode` field admits `"zk"` and the `attestation` field is opaque bytes. Migration is therefore additive and does not require protocol-level breaking changes.

7.3 Attestation registry

Verifiable Mode runtimes register their measurements (and the manufacturer attestation roots they support) in an on-chain `AttestationRegistry` contract. This allows third-party verifiers to discover the canonical mapping from runtime measurement to declared protocol version without out-of-band coordination.

8. On-Chain Registries

The protocol is bound together by a small set of on-chain registry contracts deployed on Base.

8.1 Persona contract

`Persona` is an `ERC-721` contract with the additional methods:

```
function mint(
  address to,
  bytes32 personaHash,
  string schemaVersion,
  uint16 royaltyBps
) external returns (uint256 tokenId);

function personaHash(uint256 tokenId)
  external view returns (bytes32);

function schemaVersion(uint256 tokenId)
  external view returns (string);
```

8.2 Skill Registry

`SkillRegistry` indexes all published skill manifests:

```
function publish(
  bytes32 manifestHash,
  string  id,
  string  version,
  string  category,
  uint8   riskClass
) external;

function manifest(bytes32 manifestHash)
  external view returns (
    address publisher,
    string id,
    string version,
    uint64 publishedAt
  );

function latest(string id)
  external view returns (bytes32);
```

The registry enforces uniqueness of `(id, version)` pairs per publisher and is append-only: manifests cannot be removed once published.

8.3 Composition contract

`Composition` is a separate `ERC-721` contract for explicit (persisted) compositions:

```
function compose(
  uint256 personaId,
  bytes32[] skillHashes,
  bytes32 runtimeParamsHash
) external returns (uint256 compositionId);
```

A composition references a persona by token ID and skills by content hash. Note that the composition contract does not enforce that the caller owns the referenced persona; compositions may be created over any public persona, with royalties (if configured) accruing to the persona owner per the persona's metadata.

8.4 Attestation Registry

`AttestationRegistry` indexes registered verifiable-mode runtimes:

```
function register(
  bytes32 runtimeMeasurement,
  string  measurementType,
  bytes   attestationRoot
) external;

function isRegistered(
  bytes32 runtimeMeasurement
) external view returns (bool);
```

8.5 Network choice

The protocol contracts are deployed on Base for three reasons: (i) low transaction costs sustain the read-heavy access patterns of persona and skill registries; (ii) Base's account-abstraction support simplifies user-side signature delegation for skill calls; and (iii) Base's EVM equivalence permits the protocol to be ported to other EVM-compatible chains with minimal modification if multi-chain deployment is desired in the future.

9. Storage and Off-Chain State

9.1 Content storage

Persona payloads and skill manifests are stored on IPFS, with content identifiers (CIDv1) referenced from the on-chain registries through their keccak256 hashes. For long-term durability, payloads may additionally be mirrored to Arweave; the protocol does not enforce mirroring but recommends it for personas intended to outlive any single pinning service.

9.2 Execution record storage

Execution records produced in Cloud Mode are stored at the runtime operator's discretion. Records produced in Verifiable Mode may be additionally anchored on-chain (storing only the record hash) to provide a public timeline; the protocol does not mandate this.

9.3 Indexer

To support efficient discovery and querying (e.g., "find all skills in the `trading` category published in the last 30 days"), the protocol assumes the existence of one or more off-chain indexers. A reference indexer specification is published separately. Indexers are not part of the trusted base of the protocol; clients may verify any indexer claim by reading directly from the on-chain registries.

10. Identity and Signature Schemes

10.1 Account model

The protocol assumes ECDSA-secp256k1 accounts compatible with the Base network. Smart contract wallets (ERC-4337 account abstraction) are supported throughout: every place the protocol verifies a signature uses ERC-1271 for contract accounts.

10.2 Skill invocation authorization

A user authorizes a skill invocation by signing a typed message (EIP-712) of the form:

```
SkillAuth {
  composition:  bytes32,
  skill_hash:   bytes32,
  input_hash:   bytes32,
  scope:        string,
  expires_at:   uint64,
  nonce:        uint256
}
```

The runtime presents this signature to the skill endpoint along with the request. Skill endpoints SHOULD verify the signature against the user's account and reject expired or replayed authorizations.

10.3 Delegated signing

For long-lived agent operation (e.g., a research persona running for an extended period), users may delegate signing authority to an ephemeral session key. Delegation is expressed as an EIP-712 `SessionGrant` signed by the user's primary account, scoped to a specific composition and a set of skill scopes, with a time bound. The session key may then sign `SkillAuth` messages within those bounds without further user intervention.

11. Schema Evolution

The protocol is versioned at multiple layers:

- Persona schema (`cleon/persona/v1`, future `v2`, ...).
- Skill schema (`cleon/skill/v1`, ...).
- System prompt template (versioned alongside persona schema).
- Execution record format (versioned independently).

Each version is registered on-chain as a content-addressed schema. Personas and skills declare the version they conform to. Runtimes must support all declared versions for the personas and skills they accept; runtimes that drop support for old versions risk excluding part of the agent population and lose those compositions to other runtimes.

11.1 Breaking vs. non-breaking changes

A change is *non-breaking* if existing payloads remain valid under the new schema (e.g., adding optional fields). A change is *breaking* if it invalidates existing payloads (e.g., renaming required fields, changing semantics of existing fields). Breaking changes require a new major version; non-breaking changes may be introduced under the same major version.

11.2 Schema deprecation

Schema versions are never removed from the on-chain registry. Older versions may be marked *deprecated* (a flag on the registry entry) but remain interpretable. Clients may warn users when interacting with deprecated personas or skills but must not refuse to operate on them.

12. Security Considerations

12.1 Prompt injection through skill outputs

A skill that returns adversarial content into the model's context can attempt prompt injection. Mitigation: the runtime SHOULD strictly separate skill output from the persona's primary instructions; skill outputs SHOULD be wrapped in non-instructional delimiters and the model SHOULD be configured to treat skill output as data, not instructions. This is a defense-in-depth measure; no mitigation is currently complete.

12.2 Skill endpoint impersonation

A malicious party could attempt to impersonate a registered skill's endpoint. Mitigation: skill endpoints are bound at the manifest level to a publisher-controlled domain (or a content-addressed compute environment, see below), and the runtime verifies endpoint identity via TLS certificate matching to the manifest's declared domain.

12.3 Content-addressed skill compute

In future versions, the protocol may support skill execution inside content-addressed compute environments (WebAssembly modules, signed container images), where the skill's code is itself hash-anchored. This eliminates the trust assumption on the skill's endpoint operator at the cost of restricting skills to deterministic, sandboxable computation.

12.4 Persona impersonation

The protocol does not prevent two personas from sharing the same display name. Clients SHOULD display the persona's on-chain identifier or creator address alongside the display name to disambiguate.

12.5 Replay protection

`SkillAuth` messages carry an explicit `nonce` and `expires_at`; skill endpoints MUST track consumed nonces per user account or accept only single-use authorizations.

12.6 Attestation forgery

Verifiable Mode attestations are only as strong as their underlying root of trust. The Attestation Registry exposes the manufacturer roots that were valid at the time a runtime registered; if those roots are later revoked or compromised, attestations issued under them remain on-chain but should be treated as untrusted by clients.

13. Reference Implementation

A reference implementation is being developed in two layers:

- **On-chain.** Solidity contracts for `Persona`, `SkillRegistry`, `Composition`, and `AttestationRegistry`, targeting Base. Foundry-based test suite. Audit anticipated prior to mainnet deployment.
- **Off-chain.** TypeScript SDK exposing typed bindings for the schemas and contract interfaces. Python SDK to follow.

The reference Cloud Mode runtime is a stateless Node.js service that resolves compositions, invokes a configured LLM provider, mediates tool calls, and produces signed execution records. The reference Verifiable Mode runtime is a port of the same runtime to an Intel TDX-enabled environment with attestation support.

> **Scope of this document.** This specification defines protocol-level data structures, calling conventions, and contract interfaces. It does not define application-layer behavior (studios, marketplaces, search, discovery) which are expected to be built by independent teams on top of the protocol.

14. Protocol Token: $CLEON

The protocol is associated with a native token, $CLEON, deployed as a standard `ERC-20` contract on Base. The token is not required to interact with the core protocol primitives specified in Sections 3–8: persona minting, skill registration, composition, and execution all function without reference to $CLEON. The token exists as a community coordination primitive layered on top of the protocol, not as a gatekeeping mechanism within it.

14.1 Supply and emission

$CLEON has a fixed total supply of **100,000,000** tokens. The supply is minted in full at deployment; there is no inflationary issuance, no mint function callable post-deployment, and no burn function reserved to the deployer. The token contract is renounced once the initial distribution transaction has settled, rendering the supply permanently fixed.

14.2 Initial allocation

The initial allocation is structured to minimize insider concentration and maximize on-chain liquidity from the moment of launch:

- **95%** — Initial liquidity. Paired against a base asset (ETH or a stablecoin) on a Base-native automated market maker. Liquidity provider tokens are locked or burned at deployment to remove the rug-pull vector.
- **5%** — Marketing and development liquidity. Reserved for Genesis Persona onboarding incentives, community campaigns, listing requirements on secondary venues, and development expenses incurred prior to mainnet contract finalization.

There is no team allocation, no investor allocation, no advisor allocation, and no vesting schedule. All tradeable

supply is, by construction, in the hands of the open market from the first block after deployment.

14.3 Design rationale

This distribution is deliberately atypical for a protocol with this technical scope. The conventional approach—reserving 15–30% for team and advisors, with multi-year vesting—is incompatible with two priorities the protocol places above ease of long-term operator funding:

1. **No privileged actor.** The protocol's core primitives are permissionless. A large team or investor allocation would introduce an off-protocol governance gravity that contradicts the on-chain registries' open-by-default semantics.

2. **Verifiable fair launch.** Because all but 5% of supply enters circulating liquidity at deployment, the distribution is fully verifiable on-chain. There are no off-chain claims (cliffs, vesting contracts requiring trust in a deployer multisig) to audit.

> **A note on funding model.** The protocol does not levy a fee on persona minting, skill registration, or execution. Long-term development is therefore expected to be sustained by independent contributors, grant programs, and the open ecosystem of builders, rather than by protocol revenue accruing to a treasury. This is consistent with the protocol's role as a substrate, not a platform.

14.4 Token utility

$CLEON is not used as a fee token, gatekeeping stake, or required currency for protocol interaction. Its utilities are intentionally limited to:

- **Community signaling** — holding $CLEON expresses alignment with the protocol's open-substrate thesis and grants social standing within the Cleon community.

- **Genesis Persona incentives** — early persona creators may receive $CLEON allocations from the marketing reserve as recognition of contribution.

- **Optional integration** — third-party applications built on the protocol may choose to incorporate $CLEON into their own economies (e.g., as a payment token for skill calls within their UI, or as a voting weight for app-specific governance). The protocol itself imposes no such requirement.

14.5 What $CLEON is not

To prevent misunderstanding, the following are explicit non-utilities:

- $CLEON is not required to mint a persona.

- $CLEON is not required to publish or call a skill.

- $CLEON is not a stake bonded against skill quality or runtime honesty.

- $CLEON does not entitle its holder to a share of any fees, since the protocol levies none.

- $CLEON does not confer on-protocol governance over registries, schemas, or contract upgrades. Schema evolution proceeds through the public, content-addressed registry mechanism described in Section 11, open to any participant.

14.6 Contract properties

The $CLEON contract is a standard `ERC-20` with the following non-default properties:

- `Ownable` pattern not used; the contract is ownerless from deployment.

- No transfer tax, no rebase mechanism, no blacklist function, no pausability.

- Liquidity pool LP tokens locked at a verifiable third-party locker for a duration of not less than five years, or alternatively burned outright at deployment. The exact mechanism is published in the deployment transaction.

- Source code verified on the Base block explorer; ABI and source available alongside the reference implementation.

14.7 Risk acknowledgment

A fair-launch distribution of this form has known properties that participants should evaluate:

- Open-market price discovery is volatile, particularly in the first weeks following deployment.

- Absence of a vesting team allocation means the protocol's continued development is contingent on independent contributors finding the protocol valuable to maintain; there is no economic incentive built into the supply schedule.

- Absence of a treasury means there is no internal fund for audits, security responses, or coordinated upgrades; these must be financed externally or by the community.

- The token has no claim on protocol revenue and should not be evaluated as if it did.

These properties are intentional, not oversights. Prospective holders should weigh them against the corresponding properties of conventional vested distributions and decide accordingly.

15. Conclusion

The Cleon Protocol specifies an open, content-addressed, versioned substrate for autonomous AI agents. By

treating personas as on-chain assets, skills as content-addressed callable modules, and compositions as runtime-bound bindings of the two, we obtain a system in which agents are portable across runtimes, composable across capabilities, and—in the limit—verifiable in their outputs.

The design is deliberately minimal. It avoids prescribing application-layer behavior, model choice, or orchestration patterns. What it provides is a substrate: a small set of typed, hash-anchored objects and the on-chain registries that bind them together. We expect that substrate to host a long tail of agents, skills, and runtimes that no single team could design in advance.

This document is a working draft (v1.0) of the Cleon Protocol Technical Specification. Schemas, contract interfaces, and execution conventions are subject to revision before mainnet deployment. Implementers should consult the published reference implementation for the canonical artifact set.