Vanta Protocol

Verifier-Directed Settlement for Deterministic Computation

Vanta Research

September 2026

Abstract

Vanta is a proposed application protocol for purchasing deterministic computation and resolving payment through verifier-approved evidence on an EVM-compatible host chain. A requester commits to executable meaning, inputs, output policy, verifier version, expiry, and funded compensation before execution begins. Independent operators perform work away from the settlement layer and submit a result commitment together with evidence bound to the exact task and beneficiary.

The design separates execution, evidence production, and value settlement. The host chain does not repeat the original workload. It enforces task state, dispatches a bounded statement to an immutable verifier adapter, and conserves escrowed value across acceptance or expiry. This enables operator specialization without granting operators protocol authority over validity.

Draft 0.1 specifies the target architecture and a narrow reference boundary: native-value escrow, four non-upgradeable contract roles, open result submission, pull-based withdrawals, and a transparent demonstration verifier. It does not announce a public deployment, audited proof backend, canonical token contract, supply plan, bridge, or production network.

Keywords: verifiable computation, proof markets, escrow settlement, EVM, operator networks, deterministic execution.

Contents

1 Thesis and problem

Remote computation creates an asymmetric verification problem. Producing an answer may require specialized hardware, private inputs, or substantial runtime, while the party purchasing that answer may lack the ability to reproduce it. Conventional services resolve this asymmetry through operator reputation,

contractual recourse, redundant execution, or trusted audit. Those approaches can be useful, but each keeps correctness tied to an institution or repeats enough of the original work to reduce the value of outsourcing.

Cryptographic proof systems provide a different settlement primitive. A prover can produce evidence that a precise relation holds between committed inputs and an output. When verification is materially cheaper than execution, a settlement contract can condition payment on that evidence instead of on the identity of the machine that submitted it.

Vanta applies this primitive to a task market. A task is not a mutable service request. It is an immutable, domain-separated statement with a selected verifier, a funded amount, an expiry, and a defined result encoding. Operators are compensated for satisfying that statement. The settlement layer needs no view into their infrastructure or internal execution trace; it needs only the committed terms and an acceptance decision from the selected adapter.

1.1 Design question

Given deterministic program P, public input x, optional private witness w, and output y, a requester wishes to purchase evidence that $P(x, w) = y$. The protocol must bind that evidence to the intended chain, deployment, task, program, inputs, result policy, and beneficiary. It must also define deadline behavior, prevent repeated resolution, preserve funds when no valid result arrives, and remain correct under adversarial transaction ordering.

1.2 Contributions

1. A canonical commitment envelope for deterministic computation tasks.
2. A versioned verifier directory that isolates proof-system differences behind a stable settlement interface.
3. A single-terminal-state escrow lifecycle with pull-based accounting and permissionless expiry.
4. An operator model that permits specialization without granting operators validity authority.
5. An explicit separation between computational correctness, output availability, and subjective usefulness.

2 Scope and principles

2.1 Goals

Vanta is designed to settle deterministic work through a small contract surface. It should remain neutral to operator identity, permit multiple evidence systems, prevent replay between tasks and deployments, preserve requester value after unresolved expiry, and emit enough state for independent indexers to reconstruct every transition.

2.2 Non-goals

The protocol does not determine creative quality, factual truth supplied by an external oracle, legal compliance, or any proposition not represented by the chosen verifier. It does not guarantee output availability merely because an output commitment is valid. It does not guarantee that operators will accept underpriced work or that private metadata remains hidden from the host chain.

Vanta is not an independent layer-one protocol. It defines no block production, peer-to-peer consensus, fork-choice rule, native mining process, or bridge. It relies on a host chain for transaction ordering, authentication, gas accounting, and finality.

2.3 Normative language

The terms **MUST**, **MUST NOT**, **SHOULD**, and **MAY** express requirements of the target architecture. They do not assert implementation or deployment. Token-aware escrow, production proofs, reserved execution, collateral, private-data transport, and delayed governance remain outside the Draft 0.1 reference boundary unless stated otherwise.

Commit before execution.
Every field capable of changing payout meaning is fixed before a result can resolve.

Version executable meaning.
Programs, encoders, verifier keys, and adapter behavior are identified by immutable digests.

Make failure refundable.
An unresolved task reaches a deterministic return path after expiry.

Expose trust explicitly.
Administrative powers, setup assumptions, and data dependencies remain visible.

3 System model

3.1 Roles

A *requester* specifies and funds a task. An *operator* discovers supported tasks and executes them. A *prover* constructs verifier-specific evidence; operator and prover may be the same party. A *submitter* publishes the output commitment and evidence while paying host-chain gas. A *verifier adapter* evaluates a version-pinned relation. An *indexer* reconstructs protocol state from contract events without authority over settlement.

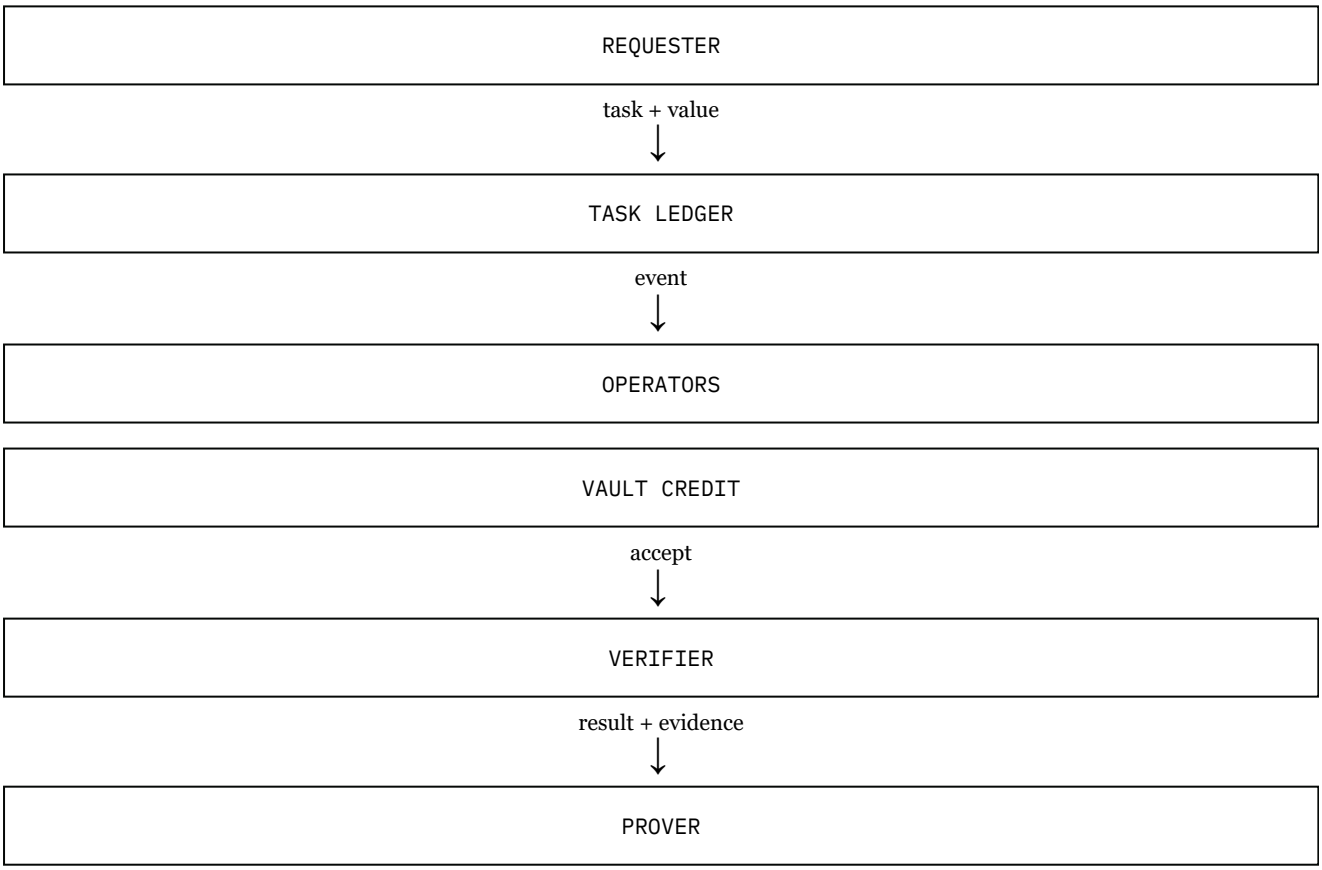

Figure 1. Execution and evidence production occur outside the host chain; acceptance and value accounting occur within the settlement contracts.

3.2 Trust assumptions

The host chain is assumed to finalize according to its published security model. The selected verifier is assumed to implement its documented relation, and the underlying evidence system is assumed sound under its stated cryptographic and setup assumptions. No operator honesty is assumed. Operators may return malformed outputs, withhold data, coordinate identities, copy calldata, or race submissions.

3.3 Notation

Symbol	Meaning
H	Canonical 256-bit commitment function.
T, c_T	Task specification and task commitment.
x, w, y	Public input, private witness, and public output.
π	Evidence consumed by the selected verifier.
D, W, C	Deposit, operator compensation, and protocol charge.
T_e	Task expiry.

4 Protocol objects

4.1 Task envelope

$$T = (\text{domain, requester, programId, inputRoot, resultPolicy, verifierId, asset, amount, expiry, mode, salt}) \tag{1}$$

The domain contains the host-chain identifier, ledger address, and major protocol version. *programId* commits to executable semantics rather than a mutable URL. *inputRoot* commits to public inputs and any encrypted input package. *resultPolicy* fixes output encoding and availability requirements. The salt separates otherwise identical requests.

$$c_T = \mathrm{H}(\mathrm{encode}(T)) \tag{2}$$

A production EVM implementation SHOULD use typed structured-data semantics or an equivalently reviewed canonical encoder. Dynamic values MUST be committed before inclusion. No off-chain component may derive task identity with an encoding different from the ledger.

4.2 Result statement

$$s = \mathrm{H}(c_T \,\|\, \text{programId} \,\|\, \text{inputRoot} \,\|\, \text{resultRoot} \,\|\, \text{beneficiary} \,\|\, \text{verifierId}) \tag{3}$$

The beneficiary is part of the verified statement so a copied transaction cannot redirect compensation. The adapter MAY include additional fields, but it MUST bind all fields required by the protocol-level statement.

4.3 Verifier record

A verifier record contains an immutable identifier, adapter address, semantic version, status, verification-key commitment, and metadata commitment. A record may be deactivated for future use, but its identifier MUST NOT be rebound to different code or keys.

5 Verification boundary

5.1 Interface

$$\text{Accept} = \text{Verify}(vk, s, \pi) \tag{4}$$

Vanta standardizes the statement boundary rather than a single proof system. Adapters may verify succinct zero-knowledge proofs, optimistic attestations with a completed dispute path, trusted execution attestations, or transparent deterministic relations. Each choice introduces different trust, setup, latency, and gas properties and therefore requires a distinct verifier identifier.

5.2 Adapter requirements

- An adapter MUST return false or revert for malformed evidence.
- It MUST bind the host domain, task, program, inputs, output, and beneficiary.
- It MUST define canonical byte encodings and reject ambiguous forms.
- Its worst-case gas usage MUST be bounded for accepted input sizes.
- Verification-key and setup assumptions MUST be published with the adapter version.

5.3 Correctness is not availability

A verifier may establish that a result root is correct without making the corresponding output bytes retrievable. Result policies that require data MUST define where those bytes appear, how they are authenticated, and how long they must remain available. Applications should not infer availability from validity alone.

6 Task lifecycle

6.1 Creation

The ledger validates task size, verifier status, duration, and funded amount against current policy. It snapshots the protocol charge and destination, transfers value to the vault, stores the task commitment, and emits the complete public envelope needed by independent operators.

6.2 Submission

A submission is eligible only while the task is open and before expiry. The ledger computes the public statement, invokes the selected adapter, and mutates state only after explicit acceptance. A verifier revert is equivalent to rejection. Implementations MUST follow checks-effects-interactions and MUST prevent reentrant terminal transitions.

6.3 Resolution

Accepted evidence records the result root and beneficiary, marks the task resolved, and instructs the vault to credit operator and protocol balances. Payment uses a pull model. The verifier path does not call recipient-controlled code.

6.4 Expiry

Once T_e has passed, any account may trigger expiry for an unresolved task. The vault credits the complete deposit to the requester. Expiry and acceptance are mutually exclusive terminal states.

```
enum TaskState { None, Open, Resolved, Refunded }

Open + accepted evidence + before expiry -> Resolved
Open + expiry reached                    -> Refunded
Resolved or Refunded                     -> no further transition
```

7 Operator market

7.1 Open execution

In open mode, any operator may perform a supported task and the first accepted beneficiary receives the compensation. This minimizes coordination state and permits immediate entry, but duplicated effort and transaction races become operator costs. Binding the beneficiary prevents theft of a public submission but does not compensate an operator whose valid work arrives second.

7.2 Reserved execution

A future reserved mode may assign a task to one operator until a reservation deadline. Assignment can reduce duplicated execution but requires discovery, bidding, liveness rules, and objective consequences for non-performance. Reserved rights MUST expire, and requester funds MUST retain a deterministic return path.

7.3 Operator selection

Operators independently evaluate program support, expected runtime, proving cost, data access, gas expense, and race probability. The protocol does not compel execution. Underpriced tasks may expire. Over time, observed resolution latency and completion rates provide public signals for price discovery without a protocol-level compute oracle.

8 Economic accounting

8.1 Conservation

$$D = W + C \tag{5}$$

The task deposit is the sole source of application-level compensation. A successful resolution credits W to the bound beneficiary and C to the charge recipient captured at creation. Expiry credits all of D to the requester. Task execution does not mint a block reward.

8.2 Prospective configuration

Policy changes apply only to tasks created after the change. A charge cap constrains governance risk. The reference target limits the charge to ten percent, though a production deployment may choose a lower cap through immutable constructor configuration.

8.3 Pull balances

Resolution credits internal balances before any external transfer. Each recipient initiates withdrawal separately. This reduces coupling between task finality and recipient behavior and makes accounting easier to test as a conservation invariant.

Outcome	Operator	Protocol	Requester
Accepted before expiry	*W*	*C*	0
Unresolved at expiry	0	0	*D*

9 Security model

9.1 Invariants

1. A task has at most one terminal outcome.
2. Withdrawn value plus claimable value never exceeds funded value.
3. Evidence accepted for one domain cannot settle another domain.
4. Evidence names the exact beneficiary credited by the vault.
5. Open-task economics and verifier identity do not mutate.
6. An unresolved expired task remains refundable without privileged cooperation.

9.2 Transaction-ordering attacks

Public evidence may be observed before inclusion. Beneficiary binding prevents redirection; private relays may reduce copying but are not a correctness dependency. A submitter can still be displaced by an earlier valid result in open mode. That is a market property rather than a settlement failure.

9.3 Administrative risk

Verifier admission and prospective limits are powerful. A production deployment SHOULD place them behind delayed multisignature control and publish every proposed change before execution. No authority should be able to replace the verifier or economics of an existing task.

9.4 Proof-system risk

Soundness failures, compromised setup material, incorrect circuits, and adapter bugs can authorize invalid settlement. Verifier versions must therefore remain independently auditable, narrowly scoped, benchmarked, and deactivatable for new tasks. Deactivation policy for already open tasks must be explicit at creation.

10 The proposed $VANTA asset

$VANTA is proposed as an optional application asset for eligible task deposits, operator payments, protocol charges, and narrowly defined collateral. It is not required for host-chain consensus and does not replace the host chain's gas asset.

This paper intentionally leaves supply, distribution, vesting, liquidity, governance, and launch timing unspecified. Those parameters require a separate document and implementation review. No address should be treated as canonical unless published through a reproducible Vanta release.

10.1 Token-aware escrow requirements

An ERC-20 vault MUST use exact balance-delta accounting, reject unsupported token behavior, isolate balances by task and asset, and preserve the same accepted-or-refunded conservation invariant as native-value escrow. Fee-on-transfer, rebasing, callback-enabled, and otherwise nonstandard assets require explicit exclusion or dedicated adapters.

11 Reference implementation

The Draft 0.1 implementation target contains four non-upgradeable EVM contracts: `TaskLedger`, `ReserveVault`, `AttestorDirectory`, and `NetworkPolicy`. A TypeScript operator indexes task events, validates an allowlist, executes a demonstration workload, constructs adapter evidence, simulates settlement, and submits a result.

The demonstration verifier recomputes a small deterministic relation directly in the EVM. Its purpose is to exercise commitments, state transitions, and accounting. It is not a production zero-knowledge backend and should not be described as one.

Capability	Draft 0.1 status	Production requirement
Native-value escrow	Reference target	Audit, invariant fuzzing, deployment manifests
Open operator mode	Reference target	Load tests, indexing resilience, reorg handling
Transparent verifier	Demonstration only	Replace with reviewed bounded evidence adapter
$VANTA settlement	Not implemented	Token specification and exact-accounting vault
Reserved execution	Not implemented	Assignment, expiry, and objective collateral design

12 Release path

Vanta should advance by narrowing uncertainty rather than expanding claims. The first milestone is a complete local path with invariant tests and reproducible state. The second is a production-grade evidence adapter with published circuits, keys, test vectors, and benchmarks. The third is a public test environment with independent operators, indexing, and adversarial exercises. Production settlement follows only after external review and operational rehearsal.

1. **Specification:** freeze encodings, state transitions, and security invariants.
2. **Reference:** exercise the full task lifecycle with intentionally transparent evidence.
3. **Proof backend:** integrate one narrow, reviewed, economically bounded relation.
4. **Public test:** operate under reorgs, congestion, malformed tasks, and hostile submitters.
5. **Audit and release:** publish source, bytecode, manifests, authority, and verified addresses.

General-purpose computation, token settlement, bridges, and governance are separate decisions. None should inherit legitimacy merely from the completion of a narrower milestone.