VEX Network

A Settlement Market for Verifiable Computation

VEX Research Group

September 7, 2026

Abstract

VEX is a research-stage market protocol for purchasing off-chain computation and settling accepted results on an EVM-compatible host chain. In the target architecture, a requester commits to a program, inputs, result policy, verifier, deadline, and escrowed reward. Workers execute outside the settlement layer and submit a result commitment with verifier-specific evidence.

The target protocol separates computation, proof generation, and financial settlement so workers and provers can specialize while the on-chain state machine remains small. A production verifier should check a bounded certificate rather than repeat the original workload. Such verification establishes only the deterministic statement encoded by that verifier, not the quality of subjective work.

This Draft 0.1 defines the target system and documents the narrower v0.1 reference implementation. The shipped code supports open native-asset jobs, four EVM contracts, one reference worker, and a transparent square verifier that recomputes arithmetic on-chain. $VEX is specified as a proposed application asset, but no public network, production contract, token contract, bridge, audited proof backend, or economic parameter set is authorized by this document.

Keywords: verifiable computation, zero-knowledge proofs, computation markets, EVM settlement, escrow.

Contents

1 Introduction

Computation is easy to request and difficult to trust. A service may return a result, but a user who cannot reproduce the work must either trust the operator, employ another operator, or accept an audit process slower than the computation itself. Cryptographic proof systems offer another primitive: a prover can produce evidence that a specified program executed over specified inputs and produced a committed output. Verification can be materially cheaper than repeating the execution.

VEX treats this evidence as the settlement condition of a market. A job is not an informal request sent to a server. It is a versioned statement with a unique commitment, a funded reward, a deadline, and an explicit verifier. A worker is paid for satisfying that statement, not for claiming that it performed work. The settlement layer therefore needs to understand neither the worker's infrastructure nor the execution trace. It checks the proof, enforces the transition, and accounts for value.

The Draft 0.1 target architecture uses an EVM-compatible host chain, with Robinhood Chain proposed for a future bounded pilot. The host supplies ordering, authentication, gas accounting, and finality. VEX supplies application-level job semantics, escrow, verifier registration, evidence acceptance, and payout. VEX is not a new consensus protocol and does not replace the host chain's security model or native gas asset.

1.1 Problem statement

Let a requester wish to evaluate deterministic program P on public input x and private witness w. The requester wants result y and evidence that $P(x, w) = y$, without relying on worker identity. The protocol must bind the proof to the exact program, inputs, result policy, chain, contract, and job. It must also decide who may submit, how long a job remains open, what happens if no proof arrives, and how payment behaves under adversarial transaction ordering.

1.2 Contributions

1. A canonical, domain-separated target commitment for deterministic compute jobs.
2. A proof-system-neutral verifier boundary with immutable verifier records.
3. A native-asset escrow implementation for open jobs and a design for future $VEX settlement and reserved execution.
4. An explicit separation between result validity, data availability, and subjective usefulness.
5. A deployment sequence beginning with one narrow, measurable workload.

1.3 Status and terminology

The key words **MUST**, **MUST NOT**, **SHOULD**, and **MAY** describe normative behavior for the target architecture. They do not imply that a feature is implemented or deployed. Unless a section explicitly says "implemented in v0.1," it is a design requirement. $VEX asset support, reserved execution, bonds, multi-asset escrow, private inputs, succinct proofs, relayers, and delayed governance are not implemented in v0.1. Example parameters are illustrative unless marked as fixed.

2 Design scope

2.1 Goals

VEX is designed to settle deterministic work through a small, auditable on-chain surface. It should be neutral to worker identity, permit multiple proving backends, prevent cross-job and cross-chain replay, preserve requester funds when no valid result arrives, and make payout conditions derivable from public state. Events must let independent indexers reconstruct every transition.

Proof-system neutrality is an interface property, not an assertion that proof systems are interchangeable. Systems differ in setup assumptions, proof sizes, verifier costs, field arithmetic, recursion, and maturity. VEX isolates those differences behind versioned verifier records while requiring each job to select one record at creation.

2.2 Non-goals

VEX does not judge creative quality, factual truth supplied by an oracle, legal compliance, or propositions not reduced to a deterministic relation. It does not conceal transaction metadata by default, guarantee acceptance of underpriced jobs, or make unavailable output data available merely because its commitment was proven correct.

VEX v0.1 is not an independent layer-one. It defines no peer-to-peer consensus, block production, fork choice, native mining process, or bridge. Architectures such as Nockchain integrate proving into base-layer consensus; VEX uses verifier acceptance as an application-level settlement condition on an existing chain.

2.3 Principles

Commit before execution.
Economically relevant fields are fixed before a result may settle.

Version executable meaning.
Programs, keys, encoders, and verifiers use immutable digests.

Make failure refundable.
A job without a valid proof reaches a deterministic refund path.

Do not hide trust.
Administrative powers, setup assumptions, and data dependencies stay explicit.

---

3 System model

3.1 Participants

A *requester* constructs and funds a job. A *worker* obtains inputs and performs execution. A *prover* produces the proof; worker and prover may be one entity. A *submitter* sends the result and proof and pays gas. A *verifier* is version-pinned code returning an acceptance decision. An *operator* may maintain discovery, storage, or proving infrastructure without protocol authority over validity.

The implemented open mode permits any address to submit the first verifier-accepted result. The target architecture also defines a reserved mode that would assign execution rights to one worker until a reservation deadline. Reservations and worker bonds are not implemented in v0.1.

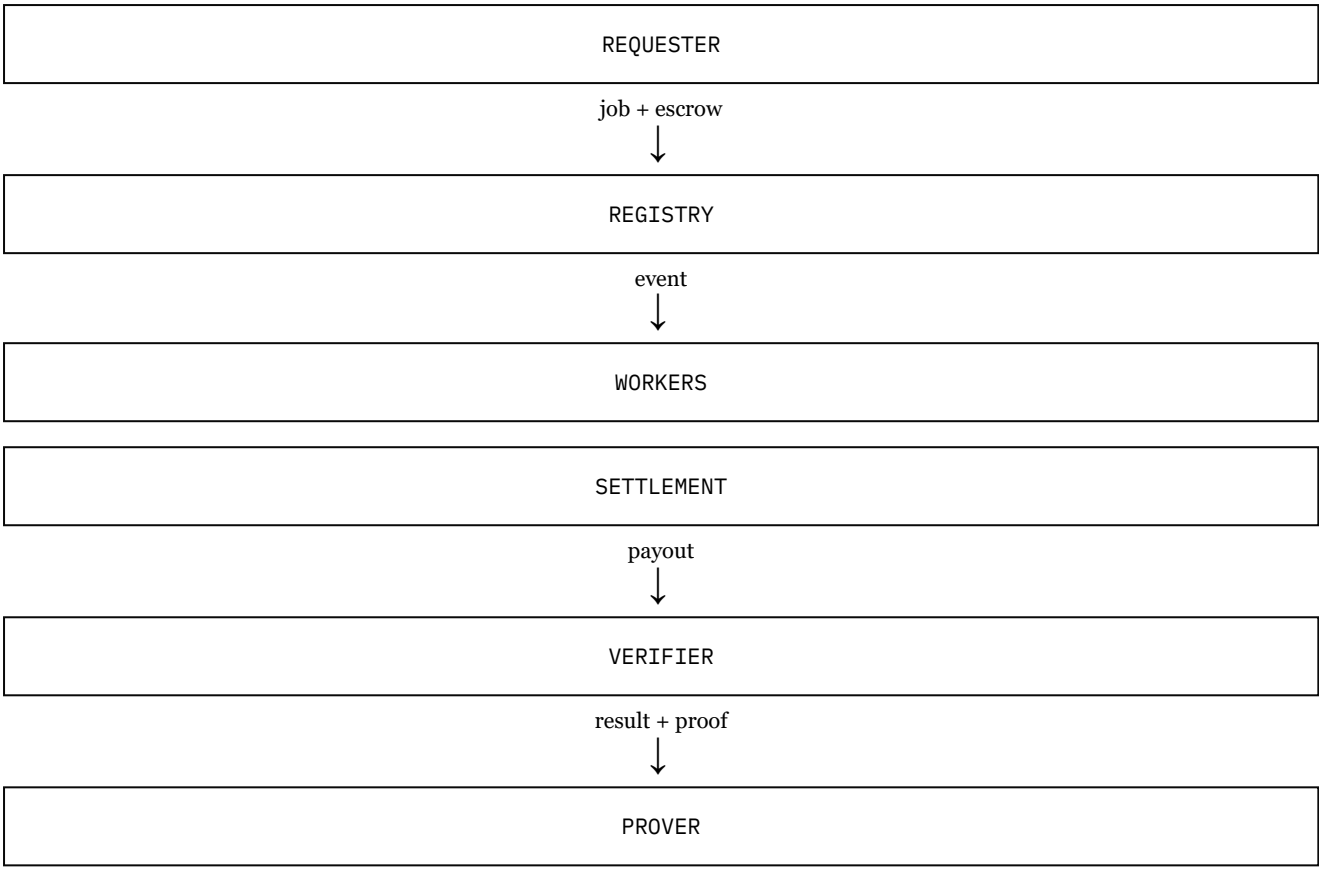

Figure 1. Computation and proving occur off-chain; proof acceptance and payment occur on the host chain.

3.2 Assumptions and adversary

The protocol assumes the host chain eventually finalizes according to its published model; the selected verifier implements the intended relation; the proof system is sound for its configured level; and output data required by the result policy can be retrieved. No honesty assumption is made about a worker. A malicious participant may withhold data, submit malformed calldata, observe the mempool, front-run public submissions, and coordinate arbitrary identities.

The adversary is assumed unable to break the commitment function, forge accepted proofs except with negligible probability, or violate finalized host-chain state. Contract behavior must remain correct when these assumptions hold and every application participant is adversarial.

3.3 Notation

Symbol	Meaning
H	Canonical 256-bit commitment function.
J, c_J	Job specification and its commitment.
x, w, y	Public input, private witness, and public result.
π	Proof accepted by the selected verifier.
R, B, F	Reward, worker bond, and protocol fee.
T_e	Job expiry.

4 Protocol objects

4.1 Job specification

The target job format is the tuple below. Fixed-width integers use unsigned interpretation and dynamic payloads use content commitments rather than ambiguous concatenation. The smaller implemented v0.1 format is documented in Appendix A.

$$J = (\text{domain}, \text{requester}, \text{programId}, \text{inputRoot}, \text{resultPolicy}, \text{verifierId}, \text{rewardAsset}, \text{rewardAmount}, \text{expir} \quad (1)$$

The domain binds chain, registry, and major protocol version. *programId* commits to executable semantics. *inputRoot* commits to public inputs and optional encrypted packages. *resultPolicy* defines output encoding and availability requirements. The salt separates otherwise identical requests.

$$c_J = \text{keccak256}(0x1901 \parallel \text{domainSeparator} \parallel \text{hashStruct}(J)) \quad (2)$$

The target EVM encoding follows EIP-712 domain separation. A production implementation MUST bind chain identifier and verifying contract. The current `jobId` uses ABI encoding rather than EIP-712 but does bind both values. A proof or signature produced for one deployment must not authorize another.

4.2 Result claim

$$C = H(c_J \parallel \text{resultRoot} \parallel \text{payoutAddress} \parallel \text{verifierId}) \quad (3)$$

In the target design, the result root commits to the canonical result and any retrieval manifest. The payout address must be inside the proven statement so a mempool observer cannot redirect payment, and a relayer may submit while beneficiary and submitter remain distinct. In v0.1, the submitter is the beneficiary and the transparent demo payload does not provide production front-running resistance.

4.3 Job states

State	Availability	Permitted exit
Open	Implemented in v0.1.	Settled or Refunded.
Settled	Implemented in v0.1.	Terminal.
Refunded	Implemented in v0.1 after deadline.	Terminal.
Reserved	Target design only.	Open, Settled, or expired refund.
Cancelled	Target design only.	Terminal after refund.

Implemented transitions are monotonic. A settled job never returns to execution, and a refund is permissionless after the deadline. Reservation and pre-deadline cancellation require a later contract version.

4.4 Events

The contract emits a complete creation event and one event for each transition. Indexers must not infer lifecycle from asset transfers. Historical replay from deployment must reconstruct the same job state as direct reads.

5 Proof interface

5.1 Verified relation

VEX mandates a relation, not one proof system. For verification key vk, public statement s, and proof π:

$$\text{Verify}(vk, s, \pi) = 1, \text{ where } s = \text{H}(c_J \parallel \text{resultRoot} \parallel \text{payoutAddress}) \quad (4)$$

Completeness requires valid executions to be accepted. Soundness requires false statements to be accepted only with negligible probability. Zero knowledge is optional: some workloads need privacy; others need only succinct integrity. Verifier metadata must disclose setup assumptions, privacy properties, and claimed security level.

5.2 Verifier registry

In v0.1, an owner-supplied verifier identifier resolves to a non-overwritable contract address, numeric version, metadata URI, and mutable active flag. Deactivation blocks both new jobs and pending settlement through that verifier. The registry does not enforce content-derived IDs or reject proxy adapters. Rich key commitments, proof-system metadata, size limits, activation blocks, and separate deprecation semantics remain target requirements.

Admission is the most sensitive governance action. A faulty verifier can authorize false work. The current registry uses a single owner and has not been audited. Production admission requires delayed governance, bytecode publication, independent audit artifacts, test vectors, and a bounded pause that does not confiscate escrow.

5.3 Adapters and aggregation

Candidate adapters include pairing-based arguments, transparent STARK systems, and zero-knowledge virtual machines. The adapter normalizes the return interface, not security assumptions. Benchmarks must report prover latency, memory, proof size, verification gas, setup, and exact software and hardware versions.

Recursive aggregation may amortize settlement but introduces another relation that must bind every child identifier and result. It is deferred from the first prototype. A future aggregator must publish an ordered child commitment list and prevent duplicate inclusion.

6 Computation market

6.1 Publication and discovery

Creation occurs on-chain; discovery may occur anywhere. A requester publishes the commitment and deposits reward plus fee. Rich metadata and input locations may be distributed by indexers, but remain advisory. If metadata conflicts with the commitment, the commitment governs.

Workers compare reward against execution, proving, transfer, failure, capital, and gas costs. The protocol does not estimate a fair on-chain price. Competing discovery markets can quote the same canonical jobs differently.

6.2 Open execution and future reservations

In implemented open mode, the first verifier-accepted submission included by the host chain wins. This minimizes coordination but may duplicate work. A production proof must bind the beneficiary to resist copied-proof redirection; the demo verifier is not a production proof system.

A future reserved mode could let a worker sign a reservation and post bond B. If it settled, the bond would be returned. If reservation lapsed, a disclosed fraction could compensate the requester and the job could reopen. No reservation or slashing logic exists in v0.1.

$$B_{\min} = \max(B_{\text{floor}}, \beta R), \quad 0 \le \beta \le 1 \quad (5)$$

6.3 Liveness, expiry, and duplicates

The v0.1 contracts guarantee refundability, not execution. If no accepted result arrives, anyone may trigger a full requester refund after T_e; there is no expiry bounty. At most one result settles per job. Quorum, streaming, and partial results require future policy adapters.

7 Settlement architecture

7.1 Contract set

JobRegistry stores canonical state. *EscrowVault* receives the native host asset and credits balances only on registry authorization. *VerifierRegistry* maps immutable verifier identifiers to implementations. *ProtocolConfig* stores bounded fee, reward, duration, input, treasury, and admission parameters applying prospectively.

Component	Authority	Forbidden behavior
JobRegistry	Create and transition jobs.	Changing verifier, reward, or expiry after creation.
EscrowVault	Move balances on registry authorization.	Arbitrary withdrawal or callbacks before state update.
VerifierRegistry	Activate or deprecate versions.	Mutating an existing verifier identifier.
ProtocolConfig	Set bounded defaults for future jobs.	Retroactively changing funded job economics.

7.2 Settlement procedure

Settlement validates state, deadline, eligibility, size bounds, and claim binding. It records the settled state and result root before any external transfer. If transfer fails, funds become a pull-based claimable balance rather than returning the job to a raceable state.

```
function settleJob(jobId, resultRoot, proof):
    job = jobs[jobId]
    require(job.state == OPEN)
    require(now <= job.expiry)
    require(verifier(job.verifierId).active)
    require(verifier.verify(job, resultRoot, msg.sender, proof))
    job.state = SETTLED
    job.resultRoot = resultRoot
    credit(msg.sender, job.reward)
    credit(job.feeRecipient, job.fee)
    emit JobSettled(jobId, msg.sender, resultRoot, job.reward, job.fee)
```

7.3 Finality, assets, and availability

Clients distinguish observed, included, and finalized settlement. Chain reorganizations are inherited from the host chain; VEX has no fork-choice mechanism. The first prototype supports only the native asset. ERC-20 support requires allowlisting because fee-on-transfer, rebasing, callback-enabled, blacklistable, and non-standard tokens can violate naive accounting.

A valid proof may attest to a correct result root while result bytes remain unavailable. Jobs requiring retrievable output must declare an availability policy. Inline output is simplest for bounded results. Larger content-addressed manifests introduce withholding and acknowledgment tradeoffs, so proof validity and bulk delivery remain explicitly separate.

8 Economics

8.1 Requester and worker cost

In v0.1, the requester sends native-asset deposit D and pays host-chain gas G_c. The configured protocol fee F is deducted from that deposit and the remainder is worker reward R.

$$D = R + F, \quad \text{Cost}_{\text{requester}} = D + G_c \tag{6}$$

The fee is snapshotted at creation and capped by contract at ten percent. It should not be presented as proof of demand. A fee schedule exceeding the value of computation drives requesters toward bilateral or centralized alternatives.

A worker compares payout against execution cost C_e, proving cost C_p, submission gas G_s, and probability q of winning an open race:

$$\mathrm{E}[\Pi] = q(R - G_s) - (C_e + C_p) \tag{7}$$

Workers reject negative expected profit regardless of nominal incentives. Benchmarking and price estimators are important market infrastructure, but remain outside consensus-critical contracts.

8.2 Conservation

$$\text{deposit} = \text{workerPayout} + \text{protocolFee}, \text{ or } \text{deposit} = \text{requesterRefund} \tag{8}$$

In v0.1, all terms are non-negative integers in the native asset's smallest unit. A multi-asset release must apply the same accounting independently in each selected asset's smallest unit. Integer fee rounding remains in the worker reward. The behavior suite checks conservation across settlement and refund paths.

8.3 $VEX application asset

$VEX is the proposed application asset of the VEX computation market. Its target economic function is to denominate eligible job deposits, worker rewards, and protocol fees. For a $VEX-denominated job, the conservation rule in Equation 8 applies in $VEX units: accepted work credits the worker and fee recipient, while expiry returns the locked deposit to the requester.

The intended relationship between computation and the asset is direct: requesters purchase verifier-accepted work, and successful workers earn the asset for satisfying the committed statement. A later reserved-execution mode may also require workers to post $VEX bonds. Any penalty must follow an objective on-chain condition, such as failure to submit before a reservation deadline after acknowledged input delivery; governance discretion alone must not be sufficient to seize a bond.

$VEX is not a native coin of the host chain. It does not pay host-chain gas, create block rewards, or participate in host-chain consensus. Requesters and submitters continue to pay gas in the host chain's native asset. This differs from an independent layer-one asset whose issuance and transaction fees are part of consensus economics.

None of this utility is implemented in v0.1, which supports native-asset escrow only. A $VEX release requires a separately reviewed token contract, exact supply and distribution specification, ERC-20 escrow support, asset admission rules, accounting invariants, deployment records, and independent audits. No contract address, supply, sale, allocation, airdrop, or launch date is established by this draft.

9 Privacy boundary

9.1 Proof privacy

A zero-knowledge backend may hide private witness w while proving a relation over public input x and result y. Whether it does so depends on the circuit, proof system, setup, and encoding. VEX cannot infer privacy because a verifier returns true; metadata must declare public and witness-only fields.

9.2 Settlement metadata

Job timing, requester, native-asset amount, verifier, expiry, beneficiary, input envelope, and transitions are public in v0.1. Stronger privacy requires encrypted delivery, privacy-preserving proofs, relays, shielded payments, private discovery, and timing protections not present in the reference implementation.

9.3 Private input delivery

A future requester may encrypt input packages to a reserved worker and commit to a ciphertext manifest. This is not implemented in v0.1, whose input envelope is emitted publicly. Any later reserved mode would need an acknowledgment window before a worker could become slashable.

9.4 Retention

On-chain commitments and events are permanent according to the host chain's retention model. Deleting an off-chain input does not erase its commitment or transaction metadata. Applications handling personal or regulated data must evaluate whether immutable publication of even a hash is acceptable.

10 Security analysis

10.1 Core invariants

1. **Escrow conservation.** No terminal path distributes more value than deposited.
2. **Single settlement.** At most one result creates a worker payout.
3. **Statement binding.** A production proof must bind one chain, registry, job, result, and beneficiary.
4. **Verifier identity.** A registered verifier address cannot be overwritten, though its active flag can change.
5. **Refund liveness.** Unsettled principal can be recovered after expiry without privilege.

10.2 Threat analysis

Threat	Consequence	Current or required control
Unsound verifier	False work receives payment.	Immutable verifier addresses and deactivation exist; audits, vectors, and delayed governance are required before a pilot.
Proof replay	One proof settles another job.	The interface supplies chain-bound job and beneficiary data; each production verifier must prove that binding.
Mempool copying	Observer submits evidence first.	Production proofs must bind the beneficiary; the demo verifier is not front-running resistant.
Reentrancy	Repeated payout or inconsistent state.	State before distribution, guarded entry, and pull balances are implemented.
Calldata denial	Excessive gas or verifier revert.	Input envelopes are bounded; proof-specific bounds remain a production requirement.
Unavailable output	Correct root but unusable result.	An explicit availability policy is a target feature, not a v0.1 guarantee.
Administrative compromise	Unsafe fees, limits, pause, or verifier admission.	Bounded fee and pause exist; delayed multisignature governance is required before a pilot.
Chain reorganization	Observed settlement disappears.	The worker uses configurable confirmations; clients still need a finality policy.

10.3 Verifier failure

A verifier bug is catastrophic for jobs selecting it. Governance cannot repair accepted false results without discretionary rollback, which VEX excludes. Containment pauses new exposure, deprecates the version, publishes the incident, and lets unsettled jobs expire or migrate only with requester authorization.

10.4 Ambiguous programs

A proof can be sound while the program is wrong. Program identifiers must commit to executable bytes, runtime semantics, field conversions, serialization, and output policy. Human-readable descriptions are not security boundaries. Reference workloads require known-answer tests and differential execution where practical.

10.5 Administrative controls

The v0.1 owner may pause new job creation and deactivate verifier records. These controls do not block expiry refunds or withdrawal of credited balances. There is no timelock or multisignature in v0.1; any public pilot must disclose and harden every privileged function and signer.

11 Reference implementation

11.1 Prototype workload

The shipped v0.1 worker supports one deterministic integration workload rather than arbitrary programs. An input envelope ABI-encodes one unsigned integer; the worker computes its square and *DemoSquareVerifier* checks the input commitment, arithmetic, job identifier, submitter, and result encoding on-chain. This exercises discovery, execution, verifier routing, escrow settlement, duplicate-settlement rejection, and expiry. Its public evidence can be recomputed for another submitter, so it is deliberately not presented as a zero-knowledge, scalable, or production front-running-resistant proof system.

11.2 Repository layout

```
contracts/         four protocol contracts and verifier interface
contracts/demo/    non-production integration verifier
scripts/           local deployment and demo-job creation
test/              contract behavior tests
node/              TypeScript worker and Dockerfile
RUN_A_NODE.md       operator guide
docs/, whitepaper/  public documentation and web paper
index.html          public site entry point
```

11.3 Encoding and workflow

The contract and worker share canonical ABI encoding. The input commitment is the hash of the exact envelope bytes. The result commitment binds the job identifier, submitting beneficiary, and result. Production adapters must preserve this domain separation when replacing the demonstration verifier.

```
$ npm test
$ npm run chain
$ npm run deploy:local
$ npm run job:demo

$ cd node
$ npm ci
$ npm start
```

11.4 Testing and reproducibility

The published behavior suite covers funded job creation, fee snapshots, escrow conservation, valid settlement, invalid-evidence rejection, duplicate-settlement rejection, permissionless expiry refunds, withdrawal accounting, admission pause, verifier deactivation, and input limits. A local end-to-end run deploys all contracts, emits a funded job, and has the worker compute and settle it. Fuzzing, invariant campaigns, verifier-specific vectors, fork tests, and independent audit remain release gates for a value-bearing pilot.

A production release must include its source commit, compiler versions, lockfiles, circuit digest, key digest, contract bytecode, deployment transaction, and benchmarks. Those proof artifacts do not exist for the demo verifier. A third party must be able to rebuild matching production verifier bytecode; explorer source verification alone is insufficient.

12 Governance and upgrades

12.1 Immutable interpretation

The target architecture requires funded jobs to retain their original interpretation. v0.1 prevents a verifier record from being overwritten, but it cannot stop code behind a registered proxy from changing. A production admission process must reject mutable adapters, pin immutable code and key commitments, and use a new versioned deployment for changed semantics.

12.2 Parameter classes

In v0.1, the owner controls verifier registration and activation, treasury, protocol fee, reward limits, duration limits, input size, and job-admission pause. Fee and limit changes apply prospectively because funded jobs snapshot their amounts and fee recipient. Proof limits, $VEX and other asset allowlists, bonds, relayers, and governance delays are target features.

12.3 Emergency process

Response consists of detection, bounded pause, notice, root-cause analysis, and versioned recovery proposal. The protocol stops new exposure rather than changing old obligations. No emergency function may transfer user escrow to governance.

13 Evaluation plan

The prototype measures proof latency, prover memory, proof bytes, verification gas, job-creation gas, settlement gas, indexer convergence, and time from creation to finalized payout. Results use named hardware and report median, p95, and worst observed values.

Gate	Evidence	Required outcome
Encoding	Independent SDK vectors.	Byte-for-byte agreement.
Correctness	Known-answer and adversarial tests.	No false acceptance in published suite.
Accounting	Stateful invariants.	Conservation across generated traces.
Performance	Reproducible benchmarks.	Verifier remains within host-chain bounds.
Operations	Public incident drill.	Pause, expiry, withdrawal operate as specified.
Review	Independent audits.	Critical findings resolved before pilot.

If proving is uneconomic, verification exceeds practical limits, or data delivery requires discretionary arbitration, that workload does not advance. Testnet exists to invalidate weak assumptions before assets depend on them.

14 Deployment sequence

Phase 0, published reference. Draft the threat model and publish the four contracts, worker, operator guide, and behavior suite. This is the current release.

Phase 1, proof candidate. Integrate one real proof backend, publish canonical vectors and benchmarks, and add fuzz and invariant testing. Accept no public value.

Phase 2, public testnet. Open jobs and workers using valueless assets. Publish metrics, limits, addresses, and incidents.

Phase 3, audit candidate. Freeze contract and circuit candidates, commission independent review, reproduce deployment, and resolve findings.

Phase 4, bounded pilot. Deploy on the selected host with per-job and aggregate caps. Support one workload and asset. Raise limits only after evaluation gates pass.

Generalized VM execution, recursive aggregation, multiple result policies, additional assets, and decentralized verifier governance remain later research. A value-bearing $VEX integration is a separate release gate after ERC-20 escrow support, a published supply and distribution specification, adversarial accounting tests, and independent review. None is required to test the core market thesis.

15 Conclusion

VEX proposes a narrow contract between computation and payment: define work precisely, commit before execution, prove the result, and settle under immutable rules. It does not eliminate trust by branding computation as verified. It moves trust into inspectable components: program, encoder, proof system, verifier, host chain, and availability policy.

The immediate objective is not scale but a falsifiable prototype whose accounting, statement binding, proof acceptance, and failure behavior can be reproduced independently. Only after those properties survive public testing should VEX be treated as more than a research specification.

R

References

1. S. Goldwasser, S. Micali, and C. Rackoff. “The Knowledge Complexity of Interactive Proof Systems.” *SIAM Journal on Computing*, 1989.
2. J. Groth. “On the Size of Pairing-based Non-interactive Arguments.” *EUROCRYPT*, 2016. eprint.iacr.org/2016/260.
3. E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. “Scalable, Transparent, and Post-Quantum Secure Computational Integrity.” 2018. eprint.iacr.org/2018/046.
4. G. Wood. “Ethereum: A Secure Decentralised Generalised Transaction Ledger.” Ethereum Yellow Paper.
5. R. Bloemen, L. Logvinov, and J. Evans. “EIP-712: Typed Structured Data Hashing and Signing.” 2017. EIP-712.
6. Nockchain Foundation. “Nockchain: A Distributed Market for Verifiable Computation.” 2026. nockchain.org/nockchain.pdf.

A

v0.1 job encoding

The deployed bytecode, not this explanatory excerpt, is normative. v0.1 creates a job from *programId*, *verifierId*, *deadline*, the exact *inputEnvelope* bytes, and native-asset *msg.value*. The contract derives:

```
nonce = nonces[msg.sender]++

jobId = keccak256(abi.encode(
    address(this),
    block.chainid,
    msg.sender,
    nonce,
    programId,
    verifierId
))

inputRoot = keccak256(inputEnvelope)
fee = msg.value * protocolFeeBps / 10_000
reward = msg.value - fee
```

The stored record also snapshots requester, fee recipient, creation time, deadline, reward, and fee. The target tuple in Section 4 is deliberately richer and requires a later version plus published compatibility vectors.

B

Implemented v0.1 job interface

```
interface IVEXJobsV01 {
    function createJob(
        bytes32 programId,
        bytes32 verifierId,
        uint64 deadline,
        bytes calldata inputEnvelope
    ) external payable returns (bytes32 jobId);

    function settleJob(
        bytes32 jobId,
        bytes32 resultRoot,
        bytes calldata proof
    ) external;

    function refundExpiredJob(bytes32 jobId) external;
    function getJob(bytes32 jobId)
        external view returns (Job memory);
}
```

`EscrowVault.withdraw(address payable recipient)` is a separate pull-payment call. Integrators must use the compiled ABI in the repository; this excerpt omits tuple and event declarations for readability.