

Vaeltrix Technical Specification

Oct 11, 2026 · @david

1. Executive summary

Vaeltrix is an open Soroban protocol that lets Stellar payment providers stake collateral against public, machine-readable service-level agreements, and pays affected counterparties automatically when a payment breaches its SLA.

Anchors, PSPs and regulated fintechs register on Vaeltrix, commit to one or more standard SLA templates (for example: inbound SEP-31 USDC payments from US to NG complete within 60 minutes), and lock stake as a performance bond. Every payment processed under a commitment carries an SLA reference. Independent observers measure each payment against counterparty-signed start and completion timestamps. When a payment breaches its target, a claim is filed, attested, and settled from the provider's stake without manual arbitration. Every outcome feeds a public scoreboard of reliability per provider, corridor and asset.

The result: reliability on Stellar stops being a marketing claim and becomes an enforceable, priced, queryable property of each provider.

What this document covers. The full protocol design: participants, data model, the measurement model (the hardest problem in the system), contract interfaces, claim lifecycle, oracle network, economics, threat model, SEP integration, SDK surface, and a phased build plan.

Status. Draft v0.1. Protocol name Vaeltrix (formerly the working name SLAx). All contract interfaces are proposals and will change during implementation.

Property	V0 decision
Enforced metric	Per-payment time to completion
Payment flow	SEP-31 inbound to a receiving anchor
Penalty model	Flat basis points of payment amount, capped per payment
Evidence	Dual-signed timestamps from sending and receiving sides

Property	V0 decision
Resolution	Optimistic: claim, challenge window, oracle quorum on dispute
Stake	Locked, never rehypothecated, exit cooldown longer than the claim window

2. Problem and motivation

Stellar's payment rails are fast at the ledger layer, but the user experience is set by the off-chain legs anchors operate, and nobody can see or enforce how well those legs perform.

2.1 What is broken today

- **Performance varies widely and invisibly.** Anchors differ in payout latency, banking partner uptime, liquidity planning and failure handling. None of this is published in a comparable form.
- **SLAs are private.** Where service terms exist, they sit in bilateral legal agreements. A wallet integrating ten anchors negotiates ten contracts and still has no live signal.
- **No automatic remedy.** When a payment stalls, the sender's support team opens a ticket with the anchor. Resolution is manual, slow, and rarely compensates anyone.
- **Reputation is anecdotal.** Routing decisions are made from Discord threads, past incidents and sales calls.

2.2 Who carries the cost

Party	Cost today
Wallets and remittance apps	Support load, churn after stuck payments, no objective basis for routing
Sending anchors and PSPs	Reputational damage for failures in a partner's corridor
Institutions evaluating Stellar	No verifiable evidence that counterparties meet operational standards
End users	Delayed funds, no compensation, no visibility

2.3 Why on-chain enforcement

Off-chain SLAs fail for small payments because enforcement costs more than the claim. A Soroban contract makes enforcement nearly free: the bond is already locked, the rules are public, and payout is a function call. This makes SLAs economically viable at remittance ticket sizes for the first time.

2.4 Why now

- SEP-31 is the standard cross-border flow, so a single integration point covers most institutional volume.
- Soroban gives programmable escrow, authorization and event emission natively.
- Institutions entering Stellar ask for operational guarantees that crypto rails historically could not provide.

3. Design principles

Every design decision in this document traces back to one of these seven principles.

1. **Measure what counterparties can observe.** Start and end events are defined by what the sender side and receiver side each see and sign, never by the provider's own report alone.
2. **Enforce per payment, report statistically.** Penalties attach to individual breaches, which are provable. Percentile SLAs are derived from the per-payment record for display.

3. **Optimistic by default, adjudicated on dispute.** Most claims settle after an unchallenged window. Oracle quorum is only invoked when a provider contests.
4. **Stake is a bond, not a pool.** Each commitment's stake backs only that commitment. No rehypothecation, no yield strategies, no shared insurance pool in V0.
5. **Templates over bespoke terms.** A small set of canonical templates makes providers comparable. Custom terms fragment the scoreboard.
6. **Fair to providers.** Bank holidays, compliance holds and recipient-side errors are excluded by explicit, auditable carve-outs. A protocol that punishes legitimate behaviour will not get adopted.
7. **Minimal on-chain footprint.** Store commitments, claims and aggregates on-chain. Keep raw payment logs off-chain, anchored by hashes.

4. Participants and roles

Vaeltrix has five roles. One organisation can hold several (an anchor can be a provider on one corridor and a client on another), but each action is authorised by the key registered for that role.

Role	Who	Responsibilities	Economic exposure
Service Provider (SP)	Anchors, PSPs, fintechs, banks operating Stellar rails	Register, commit to templates, lock stake, tag payments, sign completion timestamps, respond to disputes	Stake slashed on valid claims
Client	Wallets, sending anchors, remittance apps, merchants	Route payments to SPs, require SLA coverage, sign start timestamps, file or delegate claims	Claim bond forfeited on invalid claims
Observer (oracle)	Independent infra operators, integrators, Vaeltrix reference oracle in V0	Collect signed timestamps, compute latency, publish attestations, vote on disputes	Observer bond slashed for provably false attestations

Role	Who	Responsibilities	Economic exposure
Beneficiary	The party that receives a penalty	Defined per template: the client by default, optionally the end user via the client	None
Governance	Protocol maintainers initially, moving to a council	Approve templates, set parameters, manage the observer set, upgrade contracts behind a timelock	Reputational

4.1 Service Providers

A provider is identified by a `provider_id` bound to one Stellar account (the admin key) and one or more operational signing keys. The provider's `stellar.toml` must list the Vaeltrix provider ID so clients can verify the binding to a known home domain (see section 17).

4.2 Clients

Clients do not need to register to receive penalties, but registered clients can sign start events, which strengthens evidence and is required for the dual-signed evidence mode.

4.3 Observers

Observers are the trust-sensitive role. V0 runs a single reference observer. V1 opens a permissioned set of at least five with a quorum threshold. Section 13 covers selection, bonding and slashing.

4.4 Governance

Governance controls parameters, not individual claims. It cannot move stake or override a resolved claim. All parameter changes pass through a timelock so providers can exit before a change applies to them.

5. Scope: goals and non-goals

V0 proves one thing end to end: a provider can bond a latency SLA on a SEP-31 corridor, and a breach pays the client automatically.

5.1 Goals

- A standard on-chain way for providers to publish SLAs, lock stake against them, and pay penalties automatically on verified breaches.
- A precise, counterparty-observable definition of payment start and completion.
- Public reliability metrics per provider, template, corridor and amount band: on-time rate, latency distribution, breach count, penalties paid, current stake.
- Asset and corridor agnostic contracts. Any Stellar asset can be the payment asset or the stake asset, subject to an allowlist.
- Stake that is locked but never rehypothecated or lent.
- A TypeScript SDK and reference observer that let an existing SEP-31 anchor integrate in days, not months.

5.2 Non-goals for V0

- **No FX or price guarantees.** Vaeltrix measures performance only.
- **No consensus-layer staking.** Unrelated to validator incentives.
- **No percentile enforcement.** Percentiles are displayed, not slashed.
- **No general arbitration.** Disputes are resolved by evidence rules and observer quorum, not human judgement on the merits.
- **No consumer-facing app.** The explorer is a reference tool for integrators.
- **No insurance product.** Vaeltrix provides performance bonds, not indemnity. Penalties are pre-agreed liquidated amounts, not compensation for actual loss. The legal characterisation needs counsel review per jurisdiction before mainnet.
- **No shared risk pool.** Each commitment stands alone.

5.3 Deferred to later versions

Capability	Target version
Multi-observer quorum with bonded observers	V1
Uptime and quote-honouring metrics	V1

Capability	Target version
Percentile-based statistical penalties	V2
SEP-24 and SEP-6 flows	V2
Penalty pass-through to end users	V2
Permissionless observer set	V3

6. Core concepts and data model

Five objects carry the whole protocol: Provider, SLA Template, SLA Commitment, SLA-Tagged Payment, and Claim. Templates are immutable once approved; everything else has a lifecycle.

6.1 Provider

```
#[contracttype]
pub struct Provider {
    pub id: BytesN<32>,          // sha256(home_domain || admin)
    pub admin: Address,         // controls stake and commitments
    pub signers: Vec<BytesN<32>>, // ed25519 keys that sign completion
    events
    pub home_domain: String,    // must serve stellar.toml listing this id
    pub metadata_uri: String,   // off-chain profile JSON
    pub status: ProviderStatus, // Active | Suspended | Exited
    pub registered_at: u64,
}
```

6.2 SLA Template

A template is a reusable, governance-approved definition of what is measured, the target, the penalty, and which payments are in scope.

```
#[contracttype]
pub struct SlaTemplate {
    pub id: u32,
    pub version: u32,
    pub metric: Metric,          // V0: TimeToCompletion
    pub start_event: EventKind,  // e.g. SenderDebitConfirmed
}
```

```

    pub end_event: EventKind,          // e.g. ReceiverCreditConfirmed
    pub target_ms: u64,                // e.g. 3_600_000
    pub grace_ms: u64,                // tolerance for clock skew, e.g.
120_000
    pub penalty: PenaltyModel,
    pub scope: Scope,
    pub carve_outs: Vec<CarveOut>,    // see section 7.5
    pub claim_window_secs: u64,       // e.g. 7 days after deadline
    pub evidence_mode: EvidenceMode,  // DualSigned | ProviderSigned |
ObserverOnly
}

#[contracttype]
pub struct Scope {
    pub asset: Address,                // SAC address of the payment asset
    pub direction: Direction,         // Inbound | Outbound
    pub sender_country: Option<Symbol>, // ISO 3166-1 alpha-2
    pub receiver_country: Option<Symbol>,
    pub min_amount: i128,
    pub max_amount: i128,
    pub payout_rail: Option<Symbol>,   // e.g. BANK, MOBILE_MONEY
}

```

The JSON form of the template, used by the SDK and explorer:

```

{
  "sla_template_id": 1,
  "version": 1,
  "metric": "time_to_completion",
  "start_event": "sender_debit_confirmed",
  "end_event": "receiver_credit_confirmed",
  "target_ms": 3600000,
  "grace_ms": 120000,
  "penalty": { "type": "per_breach_bps", "bps": 50, "cap": "500000000" },
  "scope": {
    "asset": "USDC:GA5Z...KZVN",
    "direction": "inbound",
    "sender_country": "US",
    "receiver_country": "NG",
    "payout_rail": "BANK",
    "min_amount": "100000000",
    "max_amount": "100000000000"
  },
},

```

```

    "carve_outs": ["receiver_bank_holiday", "compliance_hold_declared",
    "invalid_beneficiary"],
    "claim_window_secs": 604800,
    "evidence_mode": "dual_signed"
  }

```

6.3 SLA Commitment

A provider's opt-in to one template, backed by stake.

```

#[contracttype]
pub struct Commitment {
    pub id: u64,
    pub provider_id: BytesN<32>,
    pub template_id: u32,
    pub stake_asset: Address,
    pub stake_total: i128,
    pub stake_reserved: i128,          // held against open claims
    pub max_volume_per_period: i128, // optional coverage ceiling
    pub period_secs: u64,
    pub effective_from: u64,
    pub effective_to: Option<u64>,
    pub status: CommitmentStatus,    // Active | Paused | Exiting | Exited
    pub exit_requested_at: Option<u64>,
}

```

Available stake is `stake_total - stake_reserved`. A provider may only tag a new payment if available stake covers its maximum penalty, which guarantees every tagged payment is fully collateralised at tag time.

6.4 SLA-Tagged Payment

Payments live off-chain by default. A payment becomes visible on-chain only when it is anchored in a batch (section 14) or referenced by a claim.

```
#[contracttype]
pub struct PaymentRef {
    pub payment_id: BytesN<32>,    // sha256(sep31_transaction_id ||
    provider_id)
    pub commitment_id: u64,
    pub amount: i128,
    pub t0: u64,                    // ms, signed start
    pub t1: Option<u64>,           // ms, signed completion
    pub evidence_hash: BytesN<32>, // hash of the full evidence bundle
}
```

6.5 Claim

```
#[contracttype]
pub struct Claim {
    pub id: u64,
    pub payment: PaymentRef,
    pub claimant: Address,
    pub beneficiary: Address,
    pub penalty_amount: i128,
    pub claim_bond: i128,
    pub status: ClaimStatus,        // Filed | Challenged | Upheld | Rejected
    | Paid | Expired
    pub filed_at: u64,
    pub challenge_deadline: u64,
    pub votes_for: u32,
    pub votes_against: u32,
}
```

6.6 Identifier rules

Identifier	Derivation	Why
<code>provider_id</code>	sha256(home_domain, admin address)	Binds identity to a verifiable domain
<code>payment_id</code>	sha256(SEP-31 transaction id, provider_id)	Unique per provider, impossible to collide across anchors

Identifier	Derivation	Why
<code>commitment_id</code> , <code>claim_id</code>	Monotonic counters in contract storage	Cheap and ordered
<code>evidence_hash</code>	sha256 of canonical CBOR evidence bundle	Lets anyone verify off-chain evidence matches the claim

7. Measurement model

The protocol is only as credible as its timestamps, so Vaeltrix requires that the start of a payment is signed by the party that sent it and the completion is signed by the party that received it, never by the provider alone.

7.1 The measurement problem

In a SEP-31 inbound flow, the on-chain leg (sending anchor pays receiving anchor in USDC) is fast and publicly verifiable. The slow and failure-prone leg is the receiving anchor's off-chain payout to a bank account or mobile money wallet. That leg leaves no on-chain trace. If the receiving anchor alone reports completion time, it can backdate. If the sending side alone reports, it cannot see the payout. Vaeltrix solves this by requiring each side to sign the event it actually observes.

7.2 Canonical events

Event	Signed by	Definition	Source in SEP-31
<code>SenderDebitConfirmed</code> (t0 candidate)	Sending anchor / client	Sender's funds debited and transfer committed	Client's internal ledger
<code>OnChainSettled</code>	Stellar ledger	Payment to receiving anchor's account included in a ledger	Horizon / RPC, <code>stellar_transaction_id</code>
<code>ReceiverAccepted</code>	Receiving anchor	Receiving anchor moves status to <code>pending_receiver</code>	SEP-31 GET <code>/transactions/:id</code>
<code>ReceiverCreditConfirmed</code> (t1)	Receiving anchor, co-signed by payout rail where available	Beneficiary credited	Status <code>completed</code> , <code>completed_at</code>

Event	Signed by	Definition	Source in SEP-31
RecipientAcknowledged (optional)	End user via wallet	Recipient confirms receipt	Wallet callback

V0 default: t_0 is `OnChainSettled`, the ledger close time of the Stellar payment into the receiving anchor's account. It is objective, public, and cannot be gamed by either side. t_1 is `ReceiverCreditConfirmed`.

Using the ledger close as t_0 means the SLA measures exactly the part of the flow the receiving anchor controls, which is the fairest scope for a receiver-side SLA.

7.3 Breach condition

$$\text{latency} = t_1 - t_0 \quad \text{breach} \iff \text{latency} > \text{target_ms} + \text{grace_ms} \quad ; \text{or} \quad \big(t_1 = \text{nothing} \text{ and } \text{now} > t_0 + \text{target_ms} + \text{grace_ms}\big)$$

The second clause covers payments that never complete: if no signed completion exists by the deadline, the payment is in breach.

7.4 Evidence modes

Mode	t_0 source	t_1 source	Trust assumption	Use
DualSigned	Ledger close or client signature	Provider signature + observer attestation	Provider cannot fake t_1 without colluding observer	Default for V0
ProviderSigned	Ledger close	Provider signature only	Provider honest about t_1 ; misreporting is fraud with a signed record	Early pilots only
ObserverOnly	Observer	Observer polling SEP-31 status	Observer polling interval bounds accuracy	Fallback when a provider stops responding

Why a provider-signed t1 is still meaningful. A provider who signs a false completion time creates a signed, permanent, attributable record. If the recipient later shows the funds arrived hours after the signed time, that record is evidence of fraud. The signature converts a dispute about timing into a dispute about honesty, which providers have strong reasons to avoid.

Non-response is a breach. If the provider never signs a completion, the observer records the deadline as missed. Silence is never safer than honesty.

7.5 Carve-outs

Carve-outs pause the SLA clock. Each must be declared by the provider in real time, with a signed reason code, and each is published and counted on the scoreboard so overuse is visible.

Carve-out	Clock effect	Declaration requirement	Abuse control
<code>receiver_bank_holiday</code>	Clock pauses for the holiday window	Holiday must be in the template's published calendar	Calendar fixed per corridor, governance maintained
<code>compliance_hold_declared</code>	Clock pauses until release, max 48h	SEP-31 status moves to <code>pending_customer_info_update</code> or a declared hold code	Hold rate per provider published; template can cap at N% of volume
<code>invalid_beneficiary</code>	Payment exits SLA scope	Status <code>error</code> with reason code; refund issued	Refund must be on-chain and observable
<code>sender_info_incomplete</code>	Clock pauses until info supplied	SEP-31 <code>pending_transaction_info_update</code>	Timestamped by client response

7.6 Clock skew and time sources

- On-chain timestamps use ledger close time, accurate to roughly one ledger (about 5 seconds).
- Off-chain signatures must include an RFC 3339 timestamp; observers reject events more than 30 seconds ahead of their own NTP-synced clock.

- `grace_ms` in each template absorbs residual skew. For a 60 minute target, 2 minutes of grace is ample.

7.7 Signed event format

```
{
  "v": 1,
  "event": "receiver_credit_confirmed",
  "payment_id": "b3f1...9a",
  "sep31_tx_id": "82fhs729f63dh0v4",
  "provider_id": "7c2e...41",
  "commitment_id": 42,
  "ts": "2026-10-11T14:03:22.418Z",
  "rail_ref": "NIP-000013261011140322",
  "sig": "ed25519:..."
}
```

The `rail_ref` field carries the payout rail's own reference (for example a NIBSS NIP session ID in Nigeria). It is not verified on-chain in V0, but it gives a dispute resolver a third-party record to check.

8. SLA templates and penalty models

V0 ships three templates and one penalty model; the penalty enum is designed so later models slot in without a contract migration.

8.1 Launch template set

ID	Name	Scope	Target	Grace	Penalty	Claim window
1	Fast inbound bank payout	SEP-31 inbound, USDC, bank rail, 10 to 10,000 USDC	60 min	2 min	50 bps, cap 50 USDC	7 days
2	Same-day inbound bank payout	SEP-31 inbound, USDC, bank rail, any amount	8 h	5 min	25 bps, cap 250 USDC	14 days
3	Instant mobile money payout	SEP-31 inbound, USDC, mobile money rail, 1 to 2,000 USDC	15 min	1 min	100 bps, cap 20 USDC	7 days

All figures are proposed starting parameters for testnet and will be tuned with pilot anchors.

8.2 Penalty models

```
#[contracttype]
pub enum PenaltyModel {
    // V0
    PerBreachBps { bps: u32, cap: i128 },
    // V1
    Flat { amount: i128 },
    // V1: grows with lateness, e.g. 10 bps per 15 min late
    Graduated { bps_per_step: u32, step_ms: u64, cap: i128 },
    // V2: applied once per period when percentile target is missed
    Statistical { percentile: u32, bps_of_period_volume: u32, cap: i128 },
}
```

V0 computation:

$$\text{penalty} = \min\left(\frac{\text{amount} \times \text{bps}}{10,000}, \text{cap}\right)$$

Integers only. Division rounds down in the provider's favour; the difference is at most one stroop.

8.3 Choosing penalty size

The penalty must be large enough that breaching is costlier than investing in reliability, and small enough that an honest provider's expected annual penalties stay a small fraction of revenue. A useful check:

$$\text{expected penalty cost} = \text{volume} \times \text{breach rate} \times \frac{\text{bps}}{10,000}$$

At 1M USDC monthly volume, a 2% breach rate and 50 bps, expected penalties are 100 USDC per month. A provider earning 50 to 100 bps in fees on that volume (5,000 to 10,000 USDC) bears about 1 to 2% of revenue. That is meaningful as a signal and survivable as a cost.

8.4 Template governance

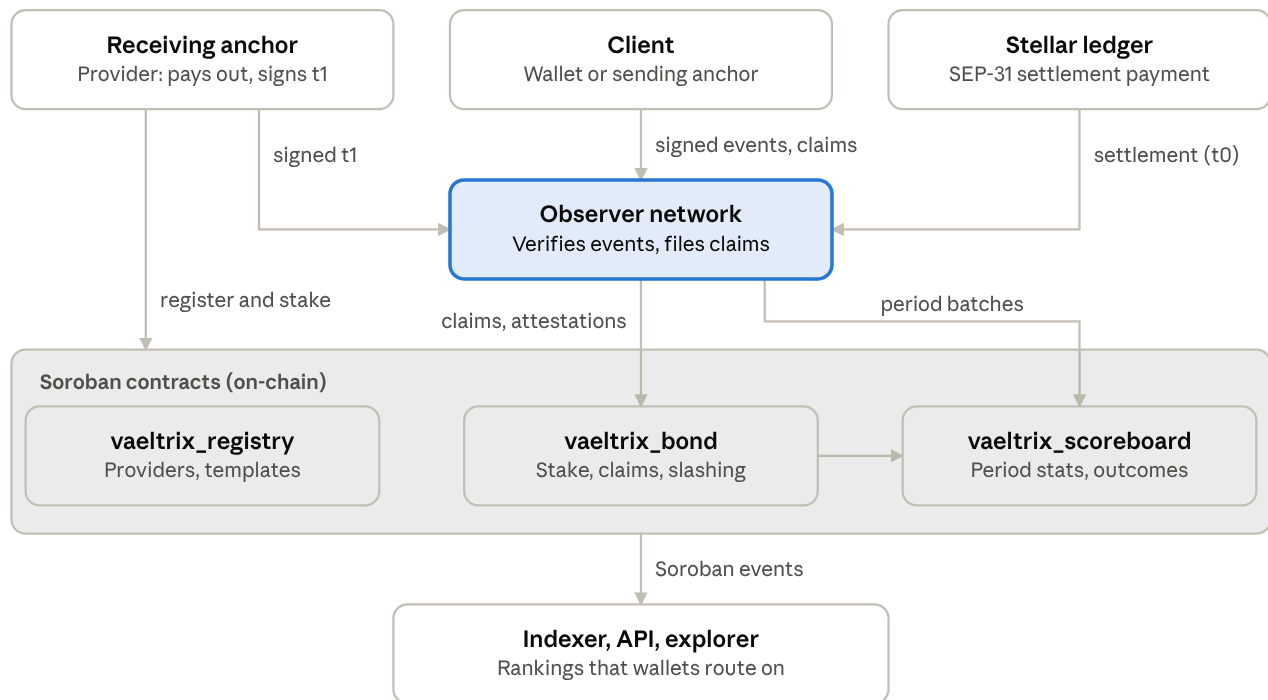
- New templates are proposed through governance and activated after a 7 day timelock.
- Templates are versioned and immutable. A change creates a new version; existing commitments stay on the version they signed.

- Corridor holiday calendars are attached to templates by reference and updated annually.

9. System architecture overview

Vaeltrix splits cleanly into an on-chain enforcement layer of three Soroban contracts and an off-chain evidence layer run by observers; the observer is the only bridge between payment reality and contract state.

Observers turn signed events into claims the contracts enforce



SLAx architecture · 3 contracts, observer network, read side

Providers and clients sign the events they observe, the ledger supplies an objective t0, observers turn evidence into claims and period batches, and an indexer exposes everything to wallets.

9.1 Layer responsibilities

Layer	Components	Holds	Trust
On-chain	<code>vaeltrix_registry</code> , <code>vaeltrix_bond</code> , <code>vaeltrix_scoreboard</code>	Identities, templates, stake, claims, aggregates	Code and governance timelock
Evidence	Observer network, signed events, evidence store	Per-payment timelines, attestations	Signatures plus observer quorum
Read side	Indexer, REST API, explorer, SDK	Derived views, scores	None: fully recomputable from the two layers above

9.2 Why three contracts

- **Registry** changes rarely and is read by everything. Keeping it separate keeps its storage cheap and its upgrade path independent.
- **Bond** holds all value. Putting claims in the same contract as stake means a slash is a single internal transfer with no cross-contract failure mode.
- **Scoreboard** accepts high-frequency batch writes. Isolating it means a bug in aggregation can never touch stake.

9.3 Trust boundaries

1. Ledger data (t0, settlement amount) is trustless.
2. Provider and client signatures are attributable but self-reported.
3. Observer attestations are the point where off-chain truth becomes on-chain fact. Section 13 covers how that trust is decentralised over time.

10. Soroban contract specifications

V0 deploys three contracts: `vaeltrix_registry` (providers and templates), `vaeltrix_bond` (commitments, stake, claims) and `vaeltrix_scoreboard` (aggregates). Claims live with stake so slashing never needs a cross-contract call that could fail midway.

10.1 vaeltrix_registry

```
pub trait RegistryInterface {
    // Providers
    fn register_provider(env: Env, admin: Address, home_domain: String,
        signers: Vec<BytesN<32>>, metadata_uri: String) ->
    BytesN<32>;
    fn update_signers(env: Env, provider_id: BytesN<32>, signers:
    Vec<BytesN<32>>);
    fn verify_domain(env: Env, provider_id: BytesN<32>); // governance or
    verifier attests toml check
    fn suspend_provider(env: Env, provider_id: BytesN<32>, reason: Symbol);
    // governance only
    fn get_provider(env: Env, provider_id: BytesN<32>) -> Provider;

    // Templates
    fn propose_template(env: Env, template: SlaTemplate) -> u32; //
    governance
    fn activate_template(env: Env, template_id: u32); // after
    timelock
    fn deprecate_template(env: Env, template_id: u32);
    fn get_template(env: Env, template_id: u32) -> SlaTemplate;

    // Observers
    fn add_observer(env: Env, observer: Address, key: BytesN<32>); //
    governance
    fn remove_observer(env: Env, observer: Address);
    fn observers(env: Env) -> Vec<Observer>;
}
```

10.2 vaeltrix_bond

```
pub trait BondInterface {
    // Commitments
    fn commit(env: Env, provider_id: BytesN<32>, template_id: u32,
        stake_asset: Address, amount: i128,
        max_volume_per_period: i128) -> u64;
    fn add_stake(env: Env, commitment_id: u64, amount: i128);
    fn request_exit(env: Env, commitment_id: u64);
    fn finalize_exit(env: Env, commitment_id: u64); // after cooldown, no
    open claims
    fn withdraw_excess(env: Env, commitment_id: u64, amount: i128); // only
```

```

above min stake
  fn pause(env: Env, commitment_id: u64);           // stops new tagging,
claims continue

  // Claims
  fn file_claim(env: Env, claimant: Address, beneficiary: Address,
                payment: PaymentRef, evidence: Bytes) -> u64;
  fn challenge(env: Env, claim_id: u64, counter_evidence: Bytes); //
provider
  fn attest(env: Env, observer: Address, claim_id: u64, uphold: bool,
            sig: BytesN<64>);
  fn resolve(env: Env, claim_id: u64); // permissionless after deadline
or quorum

  // Views
  fn commitment(env: Env, id: u64) -> Commitment;
  fn claim(env: Env, id: u64) -> Claim;
  fn available_stake(env: Env, commitment_id: u64) -> i128;
}

```

Key invariants enforced by **vaeltrix_bond**:

1. **stake_reserved** $\leq$ **stake_total** at all times.
2. A claim reserves its penalty on filing; resolution either transfers it (upheld) or releases it (rejected).
3. **finalize_exit** fails while any claim is open or while **now** $<$ **exit_requested_at** + **cooldown**.
4. **cooldown** $\geq$ **max(claim_window_secs)** + **challenge_window** + **attestation_window** for every template the commitment uses.
5. One claim per **payment_id**. A second filing for the same payment is rejected.
6. Stake tokens never leave the contract except to a beneficiary (upheld claim), to the provider (exit or excess withdrawal), or to the claimant (bond refund).

10.3 **vaeltrix_scoreboard**

```

pub trait ScoreboardInterface {
  fn anchor_batch(env: Env, observer: Address, commitment_id: u64,
                  period_start: u64, period_end: u64,
                  merkle_root: BytesN<32>, stats: PeriodStats, sig:
BytesN<64>);
  fn record_claim_outcome(env: Env, claim_id: u64); // called by

```

```

vaeltrix_bond only
  fn stats(env: Env, commitment_id: u64, period: u64) -> PeriodStats;
  fn lifetime(env: Env, provider_id: BytesN<32>) -> LifetimeStats;
}

#[contracttype]
pub struct PeriodStats {
  pub payments: u32,
  pub on_time: u32,
  pub breaches: u32,
  pub carved_out: u32,
  pub p50_ms: u64,
  pub p95_ms: u64,
  pub p99_ms: u64,
  pub volume: i128,
  pub penalties_paid: i128,
}

```

10.4 Events

Every state change emits a Soroban event so indexers never need to read storage directly.

Event topic	Data	Emitted by
prov_reg	provider_id, admin, home_domain	registry
tmpl_act	template_id, version	registry
commit	commitment_id, provider_id, template_id, stake	bond
stake_chg	commitment_id, delta, new_total	bond
claim_new	claim_id, payment_id, penalty	bond
claim_chal	claim_id	bond
claim_att	claim_id, observer, uphold	bond
claim_res	claim_id, outcome, amount	bond
exit_req / exit_fin	commitment_id	bond
batch	commitment_id, period, merkle_root	scoreboard

10.5 Storage layout

Data	Soroban storage type	Reason
Admin, parameters, counters	Instance	Small, read on every call
Providers, templates, commitments	Persistent	Long-lived, must survive archival with TTL bumps
Open claims	Persistent	Must not expire mid-dispute
Resolved claims	Temporary, TTL about 90 days	History lives in events and the indexer
Per-period stats	Persistent	Queried by wallets

The bond contract bumps TTL on every commitment and open claim it touches, and exposes a permissionless `extend_ttl(commitment_id)` for keepers.

10.6 Errors

```
#[contracterror]
pub enum Error {
    NotAuthorized = 1,
    ProviderNotActive = 2,
    TemplateNotActive = 3,
    InsufficientStake = 4,
    OutOfScope = 5,
    ClaimWindowClosed = 6,
    DuplicateClaim = 7,
    InvalidEvidence = 8,
    NotInBreach = 9,
    ChallengeWindowOpen = 10,
    OpenClaimsExist = 11,
    CooldownActive = 12,
    UnknownObserver = 13,
    AlreadyAttested = 14,
}
```

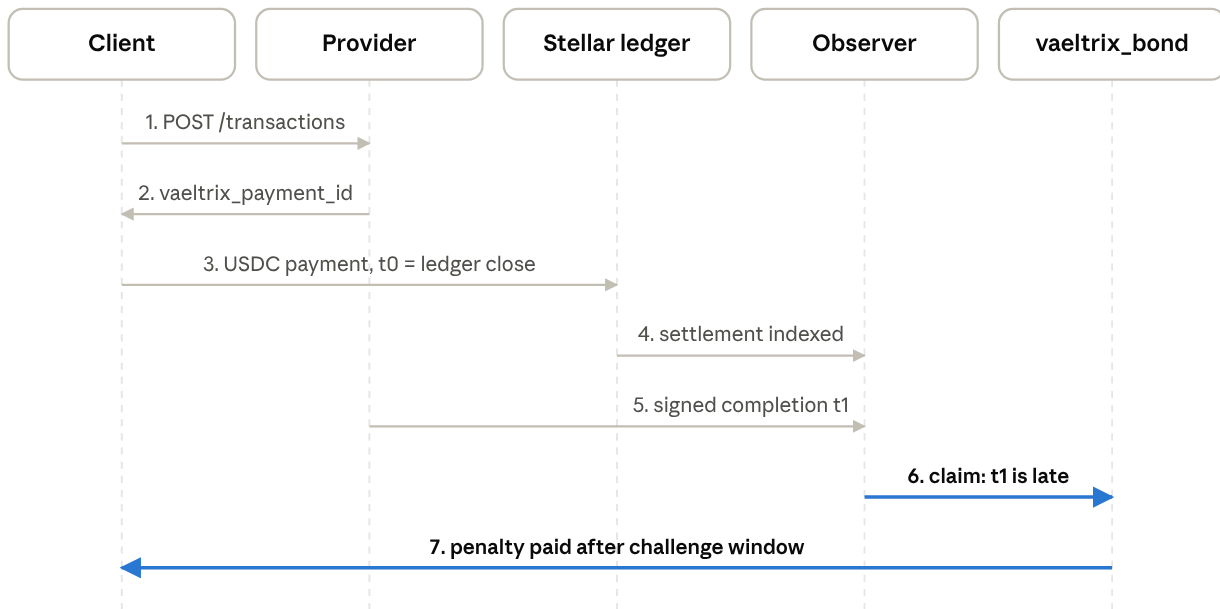
10.7 Upgradeability

All three contracts are upgradeable through `update_current_contract_wasm`, gated by governance plus a timelock of at least the longest exit cooldown. This lets any provider who disagrees with an upgrade exit with their full stake before it applies.

11. Core protocol flows

Four flows cover the protocol: onboarding, a payment under SLA, a breach claim, and exit. The breach path below is the one that matters: a late payment turns into a paid penalty without anyone opening a ticket.

A late completion becomes a penalty with no manual step



Payment under SLA · breach path, 7 messages

On the happy path, steps 6 and 7 never happen: the observer records an on-time payment in the next period batch and nothing touches stake.

11.1 Provider onboarding

1. Provider publishes the `[VAELTRIX]` block in `stellar.toml`.
2. Provider calls `vaeltrix_registry.register_provider` with admin address, home domain and signer keys.
3. Verifier fetches `stellar.toml`, confirms the IDs, and calls `verify_domain`.

4. Provider calls `vaeltrix_bond.commit` for a template, transferring stake (USDC) into the bond contract.
5. Provider adds the commitment to its SEP-31 `/info` response.
6. Observer begins indexing payments into the provider's receiving accounts.

11.2 Payment under SLA (numbered as in the diagram)

1. Client calls the provider's SEP-31 `POST /transactions` with `vaeltrix_commitment_id`.
2. Provider checks scope and confirms available stake covers the payment's maximum penalty (section 6.3), then returns `vaeltrix_payment_id`. Out-of-scope payments are accepted without coverage, never silently tagged.
3. Client sends USDC on-chain. Ledger close time becomes t_0 .
4. Observer indexes the settlement and starts the SLA clock.
5. Provider pays out and pushes a signed `ReceiverCreditConfirmed` event. If t_1 minus t_0 is within target plus grace, the payment is on time.

11.3 Breach claim

1. Observer detects breach (late t_1 , or no t_1 by deadline) and files `file_claim` on the client's behalf, or the client files directly.
2. `vaeltrix_bond` checks scope, window, duplicate and evidence hash; reserves the penalty from available stake; opens a 48 hour challenge window.
3. No challenge: anyone calls `resolve` after the window; penalty transfers to the beneficiary and the bond is refunded.
4. Challenge: section 12 applies.

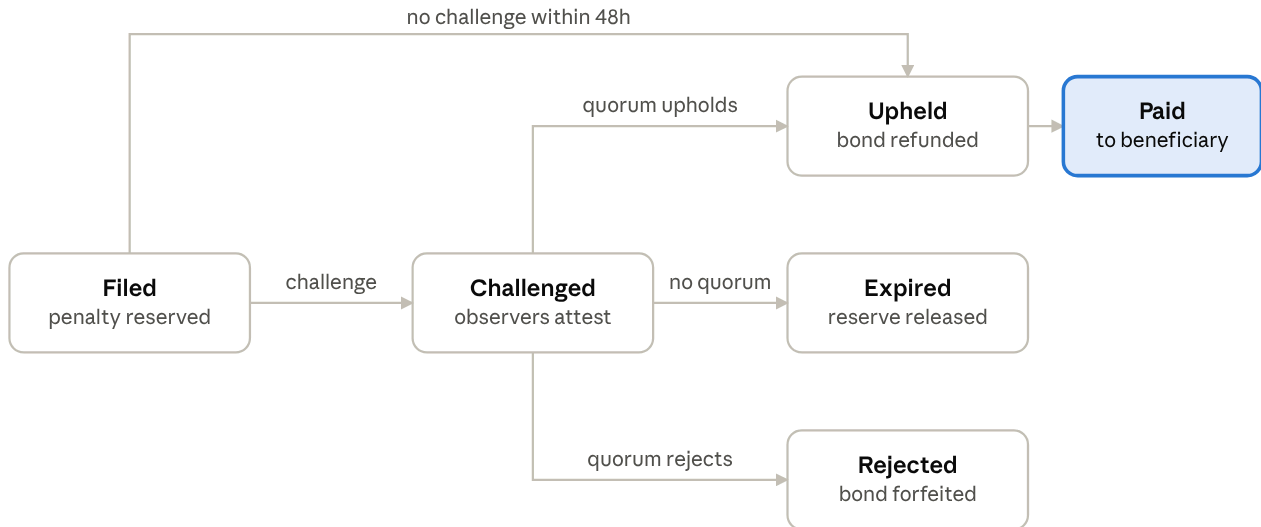
11.4 Exit

1. Provider calls `request_exit`. Commitment moves to `Exiting`; no new payments can be tagged.
2. Cooldown runs (longer than claim plus challenge plus attestation windows).
3. Provider calls `finalize_exit`. Fails if any claim is open. On success, all remaining stake returns to the provider admin.

12. Claims lifecycle and dispute resolution

Claims are optimistic: a filed claim pays out after 48 hours unless the provider challenges it, and only challenged claims reach the observer quorum.

Unchallenged claims pay out after 48 hours



Claim states · 6 states, 6 transitions

A claim that fails validation at filing (out of scope, window closed, duplicate) reverts and never enters this machine.

12.1 Validation at filing

`file_claim` reverts unless all of the following hold:

1. The commitment was `Active` at `t0`.
2. The payment amount, asset and corridor fall inside the template scope.
3. `now <= t0 + target_ms + grace_ms + claim_window_secs`.
4. No claim already exists for this `payment_id`.
5. The evidence bundle hashes to `payment.evidence_hash` and includes the ledger transaction for `t0`.
6. The breach condition (section 7.3) holds on the submitted timestamps.
7. The claimant transfers the claim bond.
8. Available stake covers the penalty; it is moved to `stake_reserved`.

12.2 Windows

Window	Proposed length	Starts at	Purpose
Claim window	7 or 14 days by template	Breach deadline	Time to detect and file
Challenge window	48 h	Filing	Provider can contest
Attestation window	72 h	Challenge	Observers vote
Exit cooldown	Sum of the above plus 7 days	Exit request	Stake cannot leave while claims are possible

12.3 Challenges

A provider challenges by submitting counter-evidence, typically a signed t1 the claimant omitted, or proof that a carve-out applied. The challenge itself is free to the provider, since the provider's stake is already at risk. Frivolous challenges show up on the scoreboard as challenge-rejection rate.

12.4 Attestation and resolution

- Each observer runs the published verdict function over the combined evidence and calls `attest` with `uphold` true or false.
- Once votes on one side reach quorum, anyone can call `resolve`.
- **Upheld:** reserved penalty transfers to the beneficiary, bond is refunded, outcome recorded on the scoreboard.
- **Rejected:** reserved penalty returns to available stake, bond is split between provider and attesting observers.
- **Expired** (no quorum within the attestation window): reserve released, bond refunded in full, claim flagged as unresolved on the scoreboard. Repeated expiries trigger an observer liveness review.

12.5 What the protocol does not judge

Vaeltrix does not decide whether a delay was fair, whether a bank was at fault, or whether the end user suffered loss. It decides only whether the signed timeline breaches the template as written. That narrowness is what makes it automatable.

13. Oracle and attestation network

Observers are the protocol's trust boundary, so Vaeltrix starts with one transparent reference observer and moves to a bonded quorum before any meaningful stake is at risk on mainnet.

13.1 What an observer does

1. Subscribes to Stellar ledgers via RPC and indexes payments into registered providers' receiving accounts. Each match produces an `OnChainSettled` event (t0).
2. Receives signed events pushed by providers and clients through the Vaeltrix event API, and polls each provider's SEP-31 `GET /transactions/:id` as a fallback.
3. Verifies signatures against the provider's registered signer keys and checks timestamps against its own clock.
4. Builds the evidence bundle for each payment and stores it, content-addressed by `evidence_hash`.
5. Files claims automatically when a breach is detected and the client has delegated filing.
6. Attests on challenged claims.
7. Publishes period batches (Merkle root of all payment records plus aggregate stats) to the scoreboard.

13.2 Phased trust model

Phase	Observer set	Quorum	Bond	Max stake per commitment
V0 testnet	1 reference observer	1 of 1	None	Testnet only

Phase	Observer set	Quorum	Bond	Max stake per commitment
V1 mainnet pilot	5 permissioned operators, at least 3 independent of the Vaeltrix team	3 of 5	Fixed per observer	Capped by governance
V2	7 to 11 operators, onboarded by governance	Two thirds	Scales with total stake secured	Raised as the set grows
V3	Permissionless with staking	Stake-weighted two thirds	Market set	Uncapped

The max-stake cap in early phases keeps the value an observer cartel could steal below the cost of corrupting it.

13.3 Attestation format

```
{
  "v": 1,
  "claim_id": 1187,
  "payment_id": "b3f1...9a",
  "evidence_hash": "e91c...07",
  "computed_latency_ms": 5512000,
  "carve_outs_applied": [],
  "verdict": "uphold",
  "observer": "GOBS...",
  "observed_at": "2026-10-11T16:12:00Z",
  "sig": "ed25519:..."
}
```

The on-chain `attest` call carries only `claim_id`, `uphold` and the signature over the canonical hash of this document. The full document is published to the observer's public log.

13.4 Observer accountability

- **Determinism.** Given the same evidence bundle and template, every honest observer must compute the same verdict. The verdict function is published as open-source code and as a test vector suite.
- **Provable misbehaviour.** An attestation that contradicts the evidence it signed over (for example, upholding a breach when signed t_1 minus t_0 is under target) is a slashable fault. Anyone can submit the evidence and the attestation to a `report_observer` function.
- **Liveness.** An observer that misses more than 10% of attestation requests in a 30 day window is flagged and can be rotated out by governance.
- **Independence.** No single organisation may run more than one observer in the V1 set. Providers cannot run observers that vote on their own claims.

13.5 Observer economics

Observers earn a fee from each resolved claim (a share of the claimant's bond on rejected claims, a share of a small protocol fee on upheld claims) plus a per-batch fee for publishing period stats. Exact split is a V1 decision; V0 has no observer fees.

14. Metrics and public scoreboard

The scoreboard is the product most users will touch: a public, verifiable reliability index for every Stellar payment provider that opts in.

14.1 Published metrics

Metric	Granularity	Source	Notes
SLA-tagged payments	Provider, template, corridor, amount band, period	Observer batches	Count and volume
On-time rate	Same	Observer batches	Excludes carved-out payments, which are shown separately
Latency p50, p95, p99	Same	Observer batches	Computed off-chain, committed via Merkle root

Metric	Granularity	Source	Notes
Breach count	Same	Claims and batches	Includes breaches nobody claimed
Carve-out rate by reason	Provider, period	Observer batches	Exposes overuse of compliance holds
Penalties paid	Provider, lifetime and period	vaeltrix_bond events	Fully on-chain
Stake at risk	Commitment	vaeltrix_bond storage	Live
Claim outcomes	Provider	vaeltrix_bond events	Upheld, rejected, expired
Coverage ratio	Commitment	Derived	Stake divided by max penalty exposure in the period

14.2 Verifiable aggregates

Computing percentiles on-chain is expensive and unnecessary. Instead, each observer publishes per period:

- a Merkle root over every payment record (`payment_id`, `amount`, `t0`, `t1`, `carve_out`, `breach`) sorted by `payment_id`,
- the aggregate `PeriodStats` signed by the observer.

Anyone can download the full record set from the observer's log, recompute the stats, check the root, and report a mismatch. In V1 each period requires matching roots from a quorum of observers before stats are marked final.

14.3 Reliability score

The explorer derives one headline number per commitment so wallets can rank providers. Proposed formula, published as a separate versioned spec so it can change without touching contracts:

```
\text{score} = 100 \times \text{on\_time\_rate}_{30d} \times \left(1 - \min\left(\left(\frac{\text{carve\_out\_rate}}{0.2}\right), 0.5\right)\right) \times \min\left(\left(\frac{\text{coverage\_ratio}}{1.5}\right), 1\right)
```

This rewards punctuality, discounts heavy carve-out use, and penalises thin collateral. A provider with fewer than 100 tagged payments in 30 days is shown as "insufficient data" rather than scored.

14.4 Privacy

Records carry no personal data: no sender or receiver names, accounts or bank details. Amounts are exact on-chain (they already are in SEP-31 settlement) but the explorer displays them in bands. Providers worried about revealing corridor volume to competitors can opt into banded volume publication, at the cost of a lower displayed confidence.

14.5 Consumption by wallets

Wallets query the scoreboard through the SDK or a public REST endpoint and can, for example, prefer the highest-scoring provider for a corridor, require a minimum score, or show users an SLA badge with the expected payout time.

15. Economic design and incentives

Vaeltrix only works if staking is profitable for good providers, so the economics are designed around a demand-side carrot (routing preference) rather than the stick alone.

15.1 Why a provider would stake

Incentive	Mechanism
Routing preference	Wallets and aggregators route to SLA-backed providers first, or only. This is the primary driver and must be secured with launch partners before mainnet.
Enterprise sales	A public, bonded SLA shortens procurement with fintechs and banks who require service terms.
Pricing power	Proven reliability justifies higher fees on premium corridors.
Lower support cost	Automatic penalties replace manual dispute handling with partners.

Incentive	Mechanism
Competitive signal	A visible score creates pressure on competitors who have not opted in.

15.2 Stake sizing

Minimum stake per commitment is set so the provider can absorb a bad day without the commitment going under-collateralised:

$$\text{min_stake} = \max\left(\text{floor}, \frac{k \times \text{daily_volume_cap} \times \text{bps}}{10,000}\right)$$

With $k = 3$ (three full days of every payment breaching). For template 1 at a 100,000 USDC daily cap and 50 bps, that is 1,500 USDC, above a proposed floor of 1,000 USDC. The real-time rule in section 6.3 (available stake must cover each new payment's max penalty) guarantees solvency regardless of the minimum.

15.3 Claimant bond

Every claim posts a bond to deter spam. Proposed value: the greater of 1 USDC or 10% of the claimed penalty.

- Claim upheld: bond refunded in full.
- Claim rejected after challenge: bond split between the provider (compensation for response cost) and attesting observers.
- Claim not challenged within 48 hours: upheld, bond refunded. Claim expired for lack of observer quorum: bond refunded in full.

15.4 Protocol fee

V0 charges no fees. A later version may take a small share of each upheld penalty to fund observers and the explorer. Any fee must be small enough that it never changes whether filing a claim is worthwhile.

15.5 Who receives the penalty

Default beneficiary is the client that sent the payment. The client decides whether to pass the penalty to the end user. V2 adds a template flag requiring pass-through, which wallets can advertise to users ("if your transfer is late, you get 0.5% back").

15.6 Stake asset

V0 accepts USDC on Stellar only, which avoids price risk on the bond and matches the payment asset in launch corridors. Accepting XLM or other assets as stake would require price feeds and haircuts and is deferred.

16. Security and threat model

The highest risks are falsified timestamps and observer collusion; contract bugs come next. Each threat below has a named mitigation and the version where it lands.

16.1 Threats

#	Threat	Actor	Impact	Mitigation	Version
T1	Provider backdates completion signature	Provider	Hides breaches, inflates score	Signed t1 is an attributable record; <code>rail_ref</code> for third-party check; recipient acknowledgement in V2; observer polling of SEP-31 status as cross-check	V0, strengthened V2
T2	Provider withholds completion until after claim window	Provider	Avoids penalty	Missing t1 at deadline is itself a breach (section 7.3)	V0
T3	Client files false claims	Client	Drains stake	Claim bond; evidence must include ledger t0; provider can challenge with signed t1	V0
T4	Spam claims to exhaust provider response capacity	Client	Griefing	Bond scales with penalty; one claim per payment; rate limits per claimant in observer	V0
T5	Provider exits with open claims	Provider	Stake escapes	Exit cooldown exceeds claim plus challenge plus attestation windows; exit blocked while claims open	V0

#	Threat	Actor	Impact	Mitigation	Version
T6	Observer colludes with provider	Observer, provider	Breaches hidden	Quorum of independent observers; client can file directly with own evidence; provably false attestation is slashable	V1
T7	Observer colludes with claimant	Observer, client	Stake stolen	Same as T6; stake caps per commitment while observer set is small	V1
T8	Wash volume: provider tags tiny or self-sent payments to pad score	Provider	Misleading metrics	Scope by amount band; per-band stats; flag payments where sender and receiver resolve to the same entity	V1
T9	Abuse of carve-outs	Provider	Hides breaches	Carve-out rate published; template-level cap on hold share	V0
T10	Signing key compromise	External	Fake completions	Multiple registered signers; rotation via admin; suspicious-pattern alerts in observer	V0
T11	Contract bug drains stake	External	Loss of funds	Audit before mainnet; invariant tests; stake caps during pilot; timelocked upgrades	Pre-mainnet
T12	Governance capture	Governance	Hostile parameter change	Timelock longer than exit cooldown; governance cannot move stake	V0
T13	State archival removes open claim	Protocol	Lost claim	Persistent storage with TTL bumps; permissionless <code>extend_ttl</code>	V0

16.2 Residual risk accepted in V0

- With a single reference observer, observer collusion (T6, T7) is mitigated only by transparency and the testnet-only scope.
- Payout leg truth ultimately depends on signatures from the provider and, optionally, the payout rail. Until rails sign directly, a determined provider willing to commit signed fraud can misreport. The protocol makes that fraud permanent and attributable, which

is the strongest guarantee available without rail integration.

16.3 Testing and assurance

- Property-based tests on all six `vaeltrix_bond` invariants (section 10.2).
- Fuzzing of claim and challenge sequences with randomised timestamps and carve-outs.
- Shared verdict-function test vectors used by both contracts and observers.
- Independent security audit of all contracts before mainnet deployment.

17. Integration with Stellar SEPs

Vaeltrix is designed to be adopted without changing SEP-31 itself: it uses existing discovery, authentication and transaction fields, and adds one optional extension that could later be proposed as its own SEP.

17.1 SEP-1 (stellar.toml)

Providers declare their Vaeltrix identity in `stellar.toml`, which binds the on-chain provider to a domain clients already trust:

```
[VAELTRIX]
PROVIDER_ID = "7c2e...41"
REGISTRY_CONTRACT = "CVAELTRIX...REG"
COMMITMENTS = [42, 57]
EVENT_SIGNING_KEYS = ["GSIGN1...", "GSIGN2..."]
```

The registry's `verify_domain` step requires a verifier to fetch this file and confirm the IDs match before the provider is marked verified.

17.2 SEP-31 (cross-border payments)

SEP-31 element	Vaeltrix use
<code>GET /info</code>	Provider adds an <code>vaeltrix</code> object per asset listing active commitment IDs and template IDs
<code>POST /transactions</code> request	Sending anchor may include <code>vaeltrix_commitment_id</code> to require coverage; receiver rejects if it cannot cover the payment

SEP-31 element	Vaeltrix use
<code>POST /transactions</code> response	Receiver returns <code>vaeltrix_payment_id</code> confirming the payment is tagged
<code>stellar_transaction_id</code>	Links the on-chain settlement to the payment, gives t0
<code>status</code> transitions	Observer polls these as a cross-check; <code>pending_customer_info_update</code> and <code>error</code> map to carve-outs
<code>completed_at</code>	Must equal the signed t1 within grace; mismatch is flagged
Callback URL	Receiver posts signed Vaeltrix events to the sender alongside standard status callbacks

Example `/info` extension:

```
{
  "receive": {
    "USDC": {
      "enabled": true,
      "vaeltrix": {
        "commitments": [
          { "commitment_id": 42, "template_id": 1, "corridors": ["US-NG"] }
        ]
      }
    }
  }
}
```

17.3 SEP-10 and SEP-45 (authentication)

The Vaeltrix event API authenticates providers and clients with SEP-10 for classic accounts and SEP-45 for contract accounts. The authenticated account must match a registered provider admin or client key.

17.4 Toward a formal SEP

Once two or more independent anchors run Vaeltrix in production, the `/info` extension, signed event format and evidence bundle should be proposed as a standalone SEP so any observer implementation can interoperate without depending on the Vaeltrix SDK.

17.5 Later flows

SEP-24 and SEP-6 deposit and withdrawal flows have similar off-chain legs and can reuse the same event model with different start events (`deposit_funds_received` , `withdrawal_payout_initiated`). These are V2 scope.

18. SDK, APIs and off-chain services

The off-chain stack is four services and one TypeScript SDK; an anchor already running SEP-31 should integrate by adding a signing hook and a few config lines.

18.1 Components

Component	Language	Responsibility
<code>@vaeltrix/sdk</code>	TypeScript	Contract bindings, event signing, evidence building, scoreboard queries
<code>vaeltrix-observer</code>	TypeScript (Node) or Rust	Ledger indexing, event ingestion, verdicts, claim filing, attestations, batches
<code>vaeltrix-indexer</code>	TypeScript + Postgres	Indexes Soroban events for the explorer and REST API
<code>vaeltrix-api</code>	TypeScript (Fastify)	Public REST: providers, commitments, stats, claims, evidence by hash
<code>vaeltrix-explorer</code>	Next.js	Provider pages, corridor rankings, claim history, contract docs

18.2 SDK surface

```
import { Vaeltrix } from "@vaeltrix/sdk";

const vaeltrix = new Vaeltrix({ network: "testnet", rpcUrl, signer });

// Provider onboarding
const providerId = await vaeltrix.providers.register({ homeDomain, signers,
metadataUri });
const commitmentId = await vaeltrix.commitments.create({
  providerId, templateId: 1, stakeAsset: USDC, amount: 1_500_0000000n,
});

// Per payment, inside the anchor's SEP-31 handler
const tag = await vaeltrix.payments.tag({ commitmentId, sep31TxId, amount });
// ...when payout completes
await vaeltrix.events.signCompletion({ paymentId: tag.paymentId, railRef, ts:
new Date() });

// Client side
const score = await vaeltrix.scoreboard.get({ providerId, corridor: "US-NG",
templateId: 1 });
const claimId = await vaeltrix.claims.file({ paymentId, beneficiary });
```

18.3 REST API (read side)

Method and path	Returns
GET /v1/providers	All providers with verification status and headline score
GET /v1/providers/:id	Profile, commitments, lifetime stats
GET /v1/commitments/:id/stats?period=30d	PeriodStats series
GET /v1/corridors/:corridor/rankings?template=1	Providers ranked by score
GET /v1/claims/:id	Claim status, attestations, evidence hash
GET /v1/evidence/:hash	Full evidence bundle
GET /v1/batches/:commitment/:period	Merkle root, stats, record download link

18.4 Event ingestion API (write side)

Method and path	Caller	Purpose
POST /v1/events	Provider or client, SEP-10 / SEP-45 authenticated	Submit a signed Vaeltrix event
POST /v1/claims/delegate	Client	Authorise the observer to file claims on the client's behalf

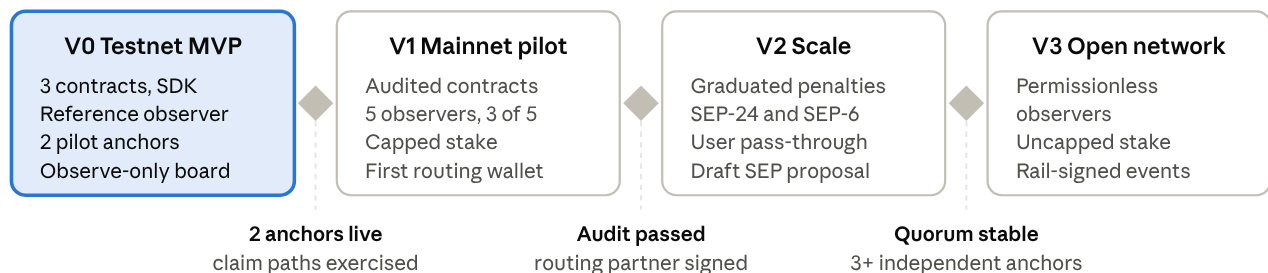
18.5 Operational requirements

- Observer uptime target 99.9%; missed events are recovered by SEP-31 polling and ledger replay.
- Evidence bundles retained for at least the longest claim window plus 1 year.
- All observer logs append-only and publicly mirrored.

19. MVP definition and phased roadmap

The MVP is a complete testnet loop with two real anchors: commit stake, tag a payment, miss the SLA, get paid. Each later phase is unlocked by a gate, not a date.

Each phase opens only when its gate is met



Roadmap · 4 phases, 3 gates, not to scale

Phases are gated by evidence of adoption and safety; dates are set only once the V0 pilot anchors are confirmed.

19.1 MVP scope

Area	In MVP	Out of MVP
Metric	Per-payment time to completion	Uptime, percentiles as penalties
Flow	SEP-31 inbound, USDC	SEP-24, SEP-6
Penalty	<code>PerBreachBps</code> with cap	Graduated, statistical
Evidence	<code>DualSigned</code> and <code>ProviderSigned</code>	Rail-signed, recipient acknowledgement
Observers	1 reference observer	Quorum
Stake asset	Testnet USDC	Other assets
Network	Testnet	Mainnet

19.2 MVP deliverables

- ☐ `vaeltrix_registry`, `vaeltrix_bond`, `vaeltrix_scoreboard` contracts with full unit and property tests
- ☐ Verdict function as a shared Rust crate, compiled into contracts and observer, with published test vectors
- ☐ `@vaeltrix/sdk` TypeScript package with contract bindings, event signing and scoreboard queries
- ☐ Reference observer: ledger indexer, SEP-31 poller, event API, auto-claim, batch publisher
- ☐ Indexer, REST API and explorer showing providers, commitments, stats and claims
- ☐ Anchor integration guide with a drop-in signing hook for common SEP-31 server setups
- ☐ Scripted demo: provider commits, client pays, provider is late, claim is filed and paid, all on testnet
- ☐ Two pilot anchors running observe-only on real testnet SEP-31 traffic

19.3 Phase gates

Gate	Criteria to pass
V0 to V1	Two anchors integrated; every claim path (upheld, rejected, expired, exit) exercised on testnet; verdict vectors stable
V1 to V2	Independent audit passed with all critical findings fixed; at least one wallet commits to routing preference; five observers onboarded
V2 to V3	Observer quorum stable over sustained operation; at least three independent anchors staked on mainnet; no unresolved observer faults

19.4 Suggested build order inside V0

1. Data types and verdict crate (shared truth for contracts and observers).
2. `vaeltrix_registry`.
3. `vaeltrix_bond`: commitments and stake first, claims second.
4. Observer ledger indexer and evidence builder.
5. Claim and challenge flows end to end.
6. `vaeltrix_scoreboard` and batch publishing.
7. SDK, API and explorer.
8. Pilot integration with two anchors.

20. Open questions, risks and glossary

The two questions that decide whether Vaeltrix succeeds are not technical: will anchors stake, and will wallets route on the score. Both must be tested with real partners before mainnet code is written.

20.1 Open questions

- ☐ Which two anchors will pilot V0, and on which corridors?
- ☐ Will at least one wallet or aggregator commit to routing preference for SLA-backed providers?

- ☐ Is a 60 minute bank payout target realistic for NG, KE and PH corridors, or should launch templates start at 4 hours?
- ☐ Can any payout rail (NIBSS NIP, M-Pesa) provide a signed or queryable completion reference for observers?
- ☐ How are penalties characterised legally in each launch jurisdiction (liquidated damages, rebate, or performance bond payout)?
- ☐ Should claims be filed automatically by observers for every breach, or only on client request?
- ☐ Who runs the first independent observers in V1?
- ☐ Should Vaeltrix be a standalone protocol or a performance module alongside an anchor risk protocol such as Sylox, sharing provider identity and data?

20.2 Key risks

Risk	Likelihood	Impact	Response
Anchors refuse to stake	High	Fatal	Lead with the free scoreboard (observation without stake), then sell bonded SLAs to providers who already score well
Wallets ignore scores	Medium	High	Secure routing commitments before mainnet; ship a wallet widget
Measurement disputes erode trust	Medium	High	Conservative carve-outs, generous grace, transparent evidence
Penalties too small to matter	Medium	Medium	Tune with pilot data; graduated penalties in V1
Regulatory reclassification as insurance	Low to medium	High	Performance bond framing, counsel review per jurisdiction, no shared pool
Observer centralisation	High in V0	Medium	Stake caps, rapid move to V1 quorum

Adoption wedge. The scoreboard can run in observe-only mode for providers who have not staked, using ledger t0 and SEP-31 polling for t1. Publishing that data creates the pressure that makes staking attractive: good providers stake to prove it, weak ones are visible either way.

20.3 Glossary

Term	Meaning
Anchor	A Stellar entity that bridges on-chain assets and fiat rails
Breach	A tagged payment whose latency exceeds target plus grace, or that never completes
Carve-out	A declared condition that pauses the SLA clock
Claim	A request to pay a penalty for a specific breached payment
Commitment	A provider's stake-backed opt-in to one SLA template
Evidence bundle	The signed events and ledger references proving a payment's timeline
Grace	Tolerance added to the target to absorb clock skew
Observer	An off-chain service that collects evidence, computes verdicts and attests
SEP-31	Stellar's standard for cross-border payments between anchors
SLA template	A governance-approved definition of metric, target, penalty and scope
t0, t1	The signed start and completion timestamps of a payment