ROOT ZERO VAULT

RSBIS/ROOT ZERO VAULT

VALIDATOR IMPLEMENTATION STACK v1.0

Complete Technical Package for Pilot Deployment

TABLE OF CONTENTS

DOCUMENT STACK INDEX

CORE DOCUMENTS (6)

6. Security & Compliance Annex v1.0
   Security architecture and compliance framework

DOCUMENT STACK INDEX

Document Stack Overview

This is a comprehensive implementation package for the RSBIS (Recursive Stage-Based Identifier System) / Root Zero Vault validator and pilot deployment.

RSBIS – A structural identity and governance protocol for high-risk digital actions and AI-era infrastructure.

Core Documents

1. Validator Overview Page

Purpose: High-level executive summary (1-2 pages)
Audience: Executives, stakeholders, slide decks, websites

Key Contents:

- "What is the Validator?" (single-sentence value proposition)
- Trust model and structural guarantees
- 8-step validation process (simplified)
- Integration pattern (POST /validate)
- Why it matters (policy as structure)
- Domain-agnostic design

Status: Complete – draft v1.0, ready for presentation

---

2. Validator Implementation Specification – v1.0

Purpose: Technical specification for building a validator service
Audience: Lead engineers, architects, security teams, implementation partners

Key Contents:

- Role and boundaries of the validator
- Core concepts (Deed, Journal, Registry, CVID, Coordinates)
- Functional requirements (deterministic, idempotent, side-effect disciplined)
- Data models (Event, Deed, Journal Entry, Registry Receipt)
- Canonicalization & CVID computation rules
- Validation flow algorithm (8-step process)
- Vault Logic execution requirements
- External API specification (HTTP with canonical YAML bodies)
- Example implementation architecture (language-agnostic)
- Logging, metrics, and testing requirements
- Deployment patterns and security considerations

Status: Complete – draft v1.0, ready for partner review

---

3. Host Entity AI Deploy Gate Implementation Guide – v1.0

Purpose: Concrete implementation guide for first pilot domain
Audience: Host-Entity implementation partner, government tech teams

Key Contents:

- Domain overview (AI Deploy Gate for a designated high-risk model)
- Complete Deed definition (EX.AI.DeployGate.Model)
    - Scope, Vault Logic, signing policy, registry policy, crypto policy
- Event & API design (request/response formats)
- Canonicalization & CVID for AI Gate domain

- Structural coordinates (A, S, T) derivation for AI deployments
    - A=1 (ModelX family), S=0/1 (staging/prod), T=sequence
- Full validation flow (10 steps)
- Turn-order enforcement logic
- Implementation checklist (14 items)

Status: Complete – draft v1.0, ready for pilot deployment

---

4. Validator Requirements for Implementation Partner (RFP-Style)

Purpose: Procurement/scoping document for hiring implementation partner
Audience: Systems integrators, large Host-Entity tech companies

Key Contents:

- Project context (Host Entity pilot domains)
- Scope of work:
    - Validator service (HTTP API, Deed loader, Vault Logic interpreter)
    - Deed & Vault Logic integration
    - Journal & Registry storage (append-only, hash-linked)
    - Integration with AI deployment pipeline
- Non-functional requirements:
    - Determinism & idempotency
    - Security (HSM, mTLS, RBAC)
    - Logging & metrics
    - Performance & availability
- Deliverables checklist
- Skills & experience expected

- RZV's role (specs, reviews, architecture support)

Status: Complete – draft v1.0, ready for RFP issuance

---

5. Validator Operations Runbook – Pilot v1.0

Purpose: Day-2 operations guide for running validator in production
Audience: DevOps/SRE teams, on-call engineers, government IT operations

Key Contents:

- Architecture snapshot (5 components)
- Environments (DEV → STAGING → PROD)
- Configuration & Deed management
    - Deed storage (versioned Git repository)
    - 5-step change process (propose, review, test DEV, test STAGING, deploy PROD)
- Deployment & rollout procedures
- Key management (generation, rotation, storage)
- Monitoring, alerts, and health metrics
- Backup & recovery (RPO/RTO, integrity verification)
- Incident response (4 standard scenarios):

1. Validator outage
2. Journal corruption/loss
3. Key compromise
4. Deed tampering

- Routine operational tasks (daily/weekly/monthly)

Status: Complete – draft v1.0, ready for operations teams

---

6. Security & Compliance Annex – v1.0

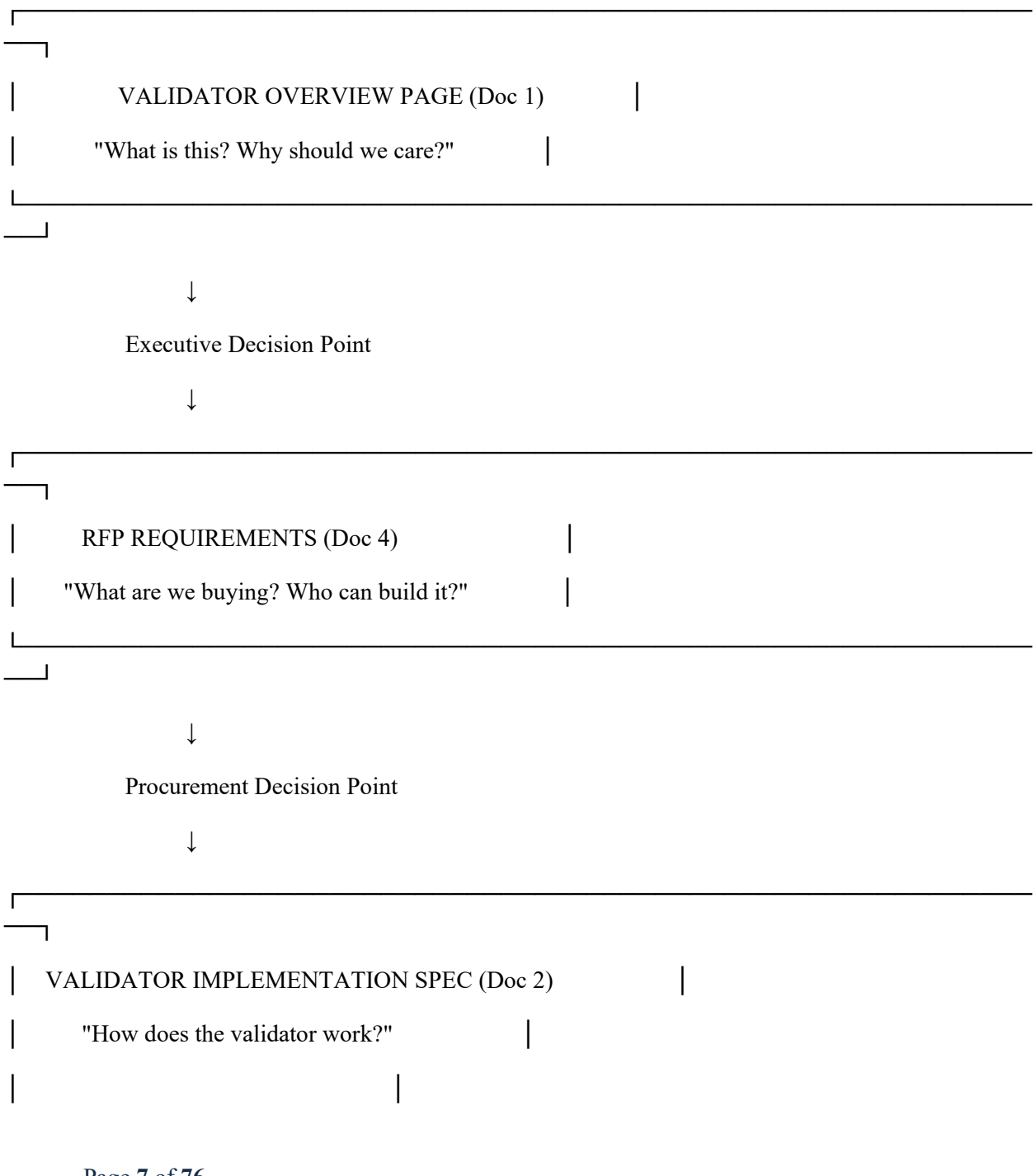
Purpose: Security architecture and compliance framework
Audience: CISO, security teams, auditors, regulators, compliance officers

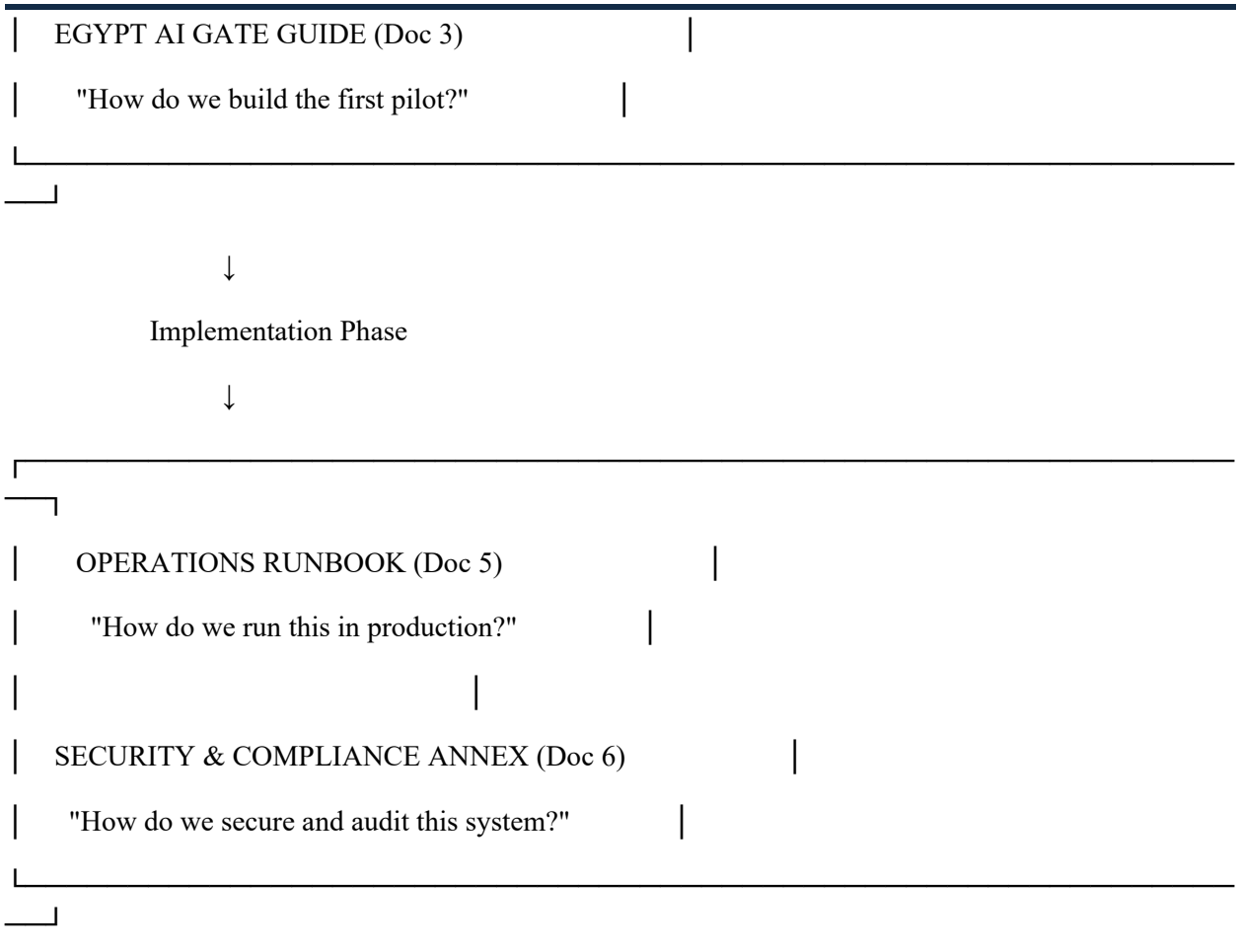
Key Contents:

- Threat model (6 primary threats with structural mitigations):
    1. Policy subversion
    2. Validator compromise
    3. Journal tampering/loss
    4. Registry poisoning
    5. Key theft/misuse
    6. Denial of service
- Identity & Access Management (5 roles with separation of duties)
- Key management (lifecycle, rotation, HSM requirements)
- Network & infrastructure security (segmentation, encryption, hardening)
- Application security (deterministic behavior, input validation, code review, supply chain)
- Evidence, audit, and integrity (Journal requirements, Registry requirements, administrative logging)
- Disaster recovery & incident response (RPO/RTO, 4 incident playbooks)
- Compliance alignment (4 domains):
    1. Governance & accountability
    2. Integrity & non-repudiation
    3. Data minimization & sovereignty
    4. AI governance & risk
- Three-phase security checklist (before/during/after pilot)

Status: Complete – draft v1.0, ready for security review

Document Relationships





Audience-to-Document Mapping

Audience	Primary Documents	Use Case
C-Suite Executives	Overview Page (1)	Decision to fund/proceed

Audience	Primary Documents	Use Case
Procurement Officers	RFP Requirements (4), Overview (1)	Tender/RFP preparation
Implementation Partner	Implementation Spec (2), Host Entity AI Gate Guide (3), RFP Requirements (4)	Building the system
DevOps/SRE Teams	Operations Runbook (5)	Day-2 operations
Security Teams	Security & Compliance Annex (6), Implementation Spec (2)	Security architecture review
Compliance/Legal	Security & Compliance Annex (6), Overview (1)	Regulatory compliance
External Auditors	Security & Compliance Annex (6), Operations Runbook (5)	Audit preparation
Government IT Leadership	All documents	Complete technical oversight

---

Key Concepts Across Documents

Deed

Definition: Mini-constitution defining scope, rules, policies for a domain
Referenced in: All documents
Example: EX.AI.DeployGate.Model (Host Entity AI Gate Guide)

Validator

Definition: Deterministic service that returns ACCEPT/REJECT decisions
Referenced in: All documents
Core Function: Execute Vault Logic, write Journal entries

Journal

Definition: Private, canonical, append-only log of all validation attempts
Referenced in: All documents
Properties: Hash-chained, encrypted at rest, tamper-evident

Registry

Definition: Minimal public receipts for accepted events
Referenced in: All documents
Properties: Subset of Journal data, queryable, interoperable

CVID (Canonical Version ID)

Definition: Cryptographic hash of canonicalized event
Referenced in: Implementation Spec (2), Host Entity AI Gate Guide (3)
Algorithm: BLAKE3 (configurable per Deed)

Coordinates (A, S, T)

Definition: Structural positioning system for events
Referenced in: Implementation Spec (2), Host Entity AI Gate Guide (3)
Components:

- A = Ancestry/lineage index
- S = Stage (lifecycle state)
- T = Turn-order (sequence number)

Vault Logic

Definition: Non-Turing, deterministic rule set in the Deed
Referenced in: All documents
Properties: Guaranteed termination, no side effects, deterministic

Fail-Closed

Definition: Default behavior when validator unavailable
Referenced in: Security Annex (6), Operations Runbook (5)
Behavior: High-risk actions are blocked, not allowed

Technical Stack Summary

Core Technologies

- Language: implementation-language agnostic (reference implementations optional)
- Hashing: BLAKE3 (configurable to SHA-256)
- Signatures: Ed25519 (configurable)
- Storage: Append-only databases (PostgreSQL + hash-chain layer recommended)
- API: HTTP REST endpoint (canonical YAML bodies)
- Configuration: YAML (Deed definitions)

Security Infrastructure

- Key Storage: HSM (Hardware Security Module) required
- Transport: mTLS (mutual TLS)
- Encryption: AES-256 at rest
- Authentication: MFA mandatory for privileged access
- Monitoring: Prometheus/Grafana (or equivalent)

Deployment Model

- Architecture: Stateless validator service behind load balancer
- Environments: DEV → STAGING → PROD progression
- Data Stores: Separate Journal (private) and Registry (public/minimal)
- Backups: Daily encrypted backups with restore testing

Pilot Implementation: Host Entity AI Deploy Gate

Domain

AI model deployment governance for high-risk a designated high-risk model

Deed ID

EX.AI.DeployGate.Model

Validation Rules

1. Dual signatures required (OpsEngineer + RiskOwner)
2. Model hash must be in approved whitelist
3. Environment must be prod or staging
4. Change ticket ID required
5. Turn-order enforced (monotonically increasing sequence)

Coordinate Mapping

- A = 1: Static for a designated high-risk model family
- S = 0: Staging environment
- S = 1: Production environment
- T: Deploy sequence number (strictly increasing)

Error Codes

- E-SIG: Signature failure
- E-AUTH: Authorization/role failure
- E-MODEL: Unapproved model hash
- E-SCOPE: Invalid environment
- E-TURN: Turn-order violation

Implementation Checklist

Phase 1: Pre-Implementation

- [ ] Review all 6 documents with implementation team
- [ ] Security team reviews Security & Compliance Annex (6)
- [ ] Legal/compliance reviews compliance sections
- [ ] Procurement issues RFP using Doc 4

- [ ] Select implementation partner

Phase 2: Implementation

- [ ] Set up DEV environment per Operations Runbook (5)
- [ ] Implement validator core per Implementation Spec (2)
- [ ] Implement Host Entity AI Gate Deed per Guide (3)
- [ ] Set up Journal & Registry storage
- [ ] Configure HSM for key management
- [ ] Implement monitoring & alerting
- [ ] Complete security checklist from Annex (6)

Phase 3: Testing

- [ ] Unit tests (all error codes)
- [ ] Integration tests (CI/CD pipeline)
- [ ] Load testing (performance baselines)
- [ ] Security testing (penetration test)
- [ ] Disaster recovery drill
- [ ] Promote to STAGING

Phase 4: Go-Live

- [ ] Final security review
- [ ] Backup & restore verification
- [ ] Monitoring dashboard operational
- [ ] On-call rotation established
- [ ] Incident playbooks accessible
- [ ] Deploy to PROD
- [ ] Monitor for 2 weeks before declaring success

Success Metrics

Technical Metrics

- Validator uptime: >99.5%
- Validation latency p95: <500ms
- Journal write success: 100%
- Hash chain integrity: No breaks
- Zero unauthorized Deed changes

Operational Metrics

- Incident response time: <15 minutes
- Backup restore success: 100%
- Key rotation completed: On schedule
- Security alerts triaged: <1 hour

Business Metrics

- AI deployments validated: 100%
- Deployment blocks (REJECTs): Justified and documented
- Audit queries answered: <24 hours
- Regulator confidence: High

Version Control

Document	Version	Date	Status
Validator Overview Page	v1.0	Draft	Ready for presentation
Validator Implementation Spec	v1.0	Draft	Ready for partner review

Document	Version	Date	Status
Host Entity AI Gate Guide	v1.0		Draft Ready for pilot deployment
RFP Requirements	v1.0		Draft Ready for RFP issuance
Operations Runbook	v1.0		Draft Ready for operations teams
Security & Compliance Annex	v1.0		Draft Ready for security review

---

Next Steps

1. Internal Review: Circulate stack to Host Entity government stakeholders
2. Security Review: CISO/security team reviews Annex (6)
3. Legal Review: Compliance team reviews regulatory sections
4. Partner Selection: Issue RFP using Doc 4
5. Pilot Kickoff: Begin implementation using Docs 2, 3, 5, 6

Contact & Governance

Document Owner: Root Zero Vault (RZV)
Pilot Owner: Host Entity – [Lead Institution TBD]
Implementation Partner: [To be selected via RFP]
Review Cycle: Quarterly during pilot, annually post-production

---

END OF INDEX

DOCUMENT 1

VALIDATOR OVERVIEW PAGE

---

1. What is the Validator?

The RSBIS Validator is a small, deterministic service that decides whether a high-risk action is allowed to happen.

It does not:

- Run AI models
- Hold citizen data
- Replace existing systems

It simply enforces:

- ACCEPT (under declared rules), or
- REJECT + reason code

...with a journaled, recomputable record that any authorized party can verify later.

For the Host Entity pilot, the first Validator governs AI deployments (the "AI Deploy Gate").

---

2. Trust Model: What the Validator Guarantees

A validator is not "just a service" – it is a structural actor with specific guarantees:

1. Determinism
   Same input + same Deed + same history ⇒ same decision and same error codes
2. Integrity
   All decisions are anchored in canonical bytes (CVIDs) and tamper-evident logs, not in mutable application state
3. Scope discipline
   A validator only evaluates events that fall under the Deeds it is configured to serve

4. Replay resistance & turn-order awareness
   Validators check prior Journal entries to enforce turn-order (T) and reject out-of-sequence or duplicate events
5. Separation of concerns
   The validator determines "may this event become true?", not "what is the business outcome?"

---

3. How It Works: 8-Step Validation Process

For every request (e.g. "deploy a designated high-risk model to production"), the Validator:

Step 1 — Loads the Deed
The Deed (e.g. EX.AI.DeployGate.Model) defines:

- Scope (what events, which environments)
- Required roles and signatures (Ops, Risk, etc.)
- Vault Logic rules (allowed models, ticket requirements, turn-order)
- What to publish in the public Registry

Step 2 — Canonicalizes & Hashes the Event (CVID)

- • Normalizes the event (lexicographically-sorted mappings, Unicode NFC)
- Computes a cryptographic hash (BLAKE3) → event_cvid
- This creates an immutable fingerprint of "what was asked"

Step 3 — Derives Structural Coordinates (A, S, T)
For the AI Gate pilot:

- A – ancestry/lineage (e.g. 1 for "a designated high-risk model family")
- S – stage (0 = staging, 1 = prod)
- T – deploy sequence (monotonically increasing counter)

Step 4 — Loads Context from the Journal

- Last accepted deploy for (A, S)
- Any prior rejected attempts (if relevant for policy)

Step 5 — Verifies Signatures & Roles

- Checks that required roles (e.g. OpsEngineer, RiskOwner) have signed
- Validates cryptographic signatures against approved keys

Step 6 — Executes Vault Logic
Example for AI Deploy Gate:

- Is the model hash in the approved list?
- Is environment one of ["prod","staging"]?
- Is there a valid change ticket ID?
- Is T > last_T for this environment (no turn-order violation)?

Step 7 — Writes to Journal & Registry

- Journal: full, encrypted, hash-chained event with decision
- Registry: minimal public receipt (hash, model version, env, ticket)

Step 8 — Returns a Signed Decision

- ACCEPT or REJECT
- Standard error code (E-SIG, E-AUTH, E-TURN, E-MODEL, …)
- References to journal_entry_id and registry_receipt_id

---

4. Integration Pattern

Simple contract:

Existing system (e.g. CI/CD pipeline) calls:

POST /validate

with the deploy request.

Validator responds with:

decision: ACCEPT

error_code: null

details: ...

The system must:

- Proceed only on ACCEPT
- Block on REJECT and display the reason

This pattern is:

- Domain-agnostic: same design for AI, Treasury, Tourism, etc.
- Infrastructure-agnostic: can run on any stack Host Entity approves
- Extensible: new Deeds = new "Doors" using the same validator pattern

---

5. Why It Matters

The Validator turns "policy on paper" into policy as structure:

- Every high-risk action is:
    - Cryptographically bound
    - Structurally checked
    - Logged as court-grade evidence

For Host Entity, this means:

- AI and financial systems can be governed architecturally, not just by procedures and memos
- It becomes possible to prove how and under which rules decisions were made, years later

---

END OF DOCUMENT 1

DOCUMENT 2

VALIDATOR IMPLEMENTATION SPECIFICATION v1.0

0. Purpose & Audience

This document defines how to implement a Validator Service for the RSBIS / Root Zero Vault architecture.

Written for:

- Lead engineers / architects
- Security / infra teams
- Implementation partners building pilots (e.g. Host Entity AI Gate, Treasury gate, Tourism IDs)

It explains:

- What a validator is and is not
- Exact inputs / outputs
- Canonicalization & CVID
- How to execute Vault Logic
- How to write to Journal and Registry
- Example API + example implementation skeleton

---

1. Role of the Validator

A Validator is a small, deterministic service whose only job is to decide:

May this event become part of canonical history under this Deed?

For each request, the validator:

1. Loads the appropriate Deed (rules, policies, crypto settings)

2. Canonicalizes the event → CVID
3. Derives structural coordinates (A, S, T; optional G)
4. Gathers relevant Journal history
5. Executes Vault Logic
6. Writes a Journal entry (always)
7. Optionally writes a Registry receipt (if accepted & required)
8. Returns a signed decision: ACCEPT or REJECT + error_code

It does not:

- Replace existing business systems
- Render UI
- Decide business outcomes (e.g. "pay invoice"); it only says if the event passes structure

---

2. Core Concepts (Quick Recap)

Deed

Mini-constitution for a domain. Contains:

- Scope (what it governs)
- Vault Logic (rules)
- Registry policy
- Signature requirements
- Crypto posture (algorithms, key types)

Journal

Private canonical log of all validation attempts (ACCEPT and REJECT). Tamper-evident.

Registry

Minimal public receipts for selected accepted events (proof without disclosure).

CVID

Cryptographic hash of the canonical event bytes. If two events are "the same", they must produce the same CVID.

Coordinates (A, S, T, G)

- A – Ancestry index / lineage
- S – Stage (state/lifecycle)
- T – Turn-order index (sequence)
- G – Optional global mixed-radix index (bijective integer for advanced uses)

Vault Logic

Non-Turing, bounded, deterministic rule set defined in the Deed. Takes event+history and returns ACCEPT or REJECT+code.

---

3. Functional Requirements

A compliant validator must:

1. Be deterministic
   Same inputs ⇒ same outputs (decision, error code, CVID)
2. Be idempotent
   Re-validating the same event (same CVID) must not create conflicting Journal records
3. Be side-effect–disciplined
   Only allowed side effects:
   - Append Journal entry
   - Append Registry receipt (if allowed by policy)
   - Emit logs / metrics
4. Be authenticatable
   Every decision must be signed by a validator key recognized by the Deed

5. Enforce scope
   Only process events for Deeds it is configured to serve

---

4. Data Model (Logical)

4.1 Event (Incoming)

Logical shape (Canonical YAML):

deed_id: EX.AI.DeployGate.Model

event_type: MODEL_DEPLOY_REQUEST

payload:

  model_hash: abc123...

  environment: prod

  requested_by: user-123

  ticket_id: IT-5621

  ts_client: 2025-10-10T13:45:00Z

signatures:

  - key_id: KEY-OPS-001

    role: OpsEngineer

    signature: base64...

  - key_id: KEY-RISK-001

    role: RiskOwner

    signature: base64...

meta:

  request_id: uuid-...

```
  source_system: ci-cd-pipeline-1
```

4.2 Deed (Simplified Logical Shape)

```
deed_id: EX.AI.DeployGate.Model
version: 1
scope:
  event_types:
    - MODEL_DEPLOY_REQUEST
vault_logic:
  model: predicate_dag
  nodes:
    - id: scope_match
      predicate:
        kind: scope_match
      on_fail:
```

```
      decision: REJECT
      error_code: E_SCOPE
    next:
      - roles_ok
  - id: roles_ok
    predicate:
      kind: all_roles_present
      roles:
        - OpsEngineer
        - RiskOwner
    on_fail:
      decision: REJECT
      error_code: E_AUTH
    next:
      - env_ok
```

4.3 Journal Entry

```
journal_entry_id: JOR-2025-10-10-000001
deed_id: EX.AI.DeployGate.Model
event_cvid: blake3:...
decision: ACCEPT
error_code: null
coords:
  A: 125
```

S: 1

T: 7

G: optional-global-index

event_snapshot: <encrypted_or_policy_minimal_snapshot>

validator_id: VAL-01

validator_signature: base64...

ts_validator: 2025-10-10T13:46:09Z

prev_journal_hash: blake3:...

4.4 Registry Receipt (If Published)

registry_receipt_id: REG-2025-10-10-000001

deed_id: EX.AI.DeployGate.Model

event_cvid: blake3:...

journal_entry_id: JOR-2025-10-10-000001

summary:

event_type: MODEL_DEPLOY_REQUEST

model_hash: abc123...

environment: prod

ts_registry: 2025-10-10T13:46:10Z

registry_hash: blake3:...

5. Canonicalization & CVID

5.1 Goals

- Two semantically identical events must yield identical CVIDs
- Trivial differences (field order, whitespace) must be normalized away
- All signers and validators must compute the same bytes before hashing

5.2 Canonicalization Rules (Recommended)

1. Use UTF-8 encoding for all text
2. Apply Unicode NFC normalization
3. 3. For structured mapping payloads:
    - Sort keys recursively in lexicographic order
    - Use a fixed canonical serializer (no pretty-print, no trailing zeros unless structural)
4. Exclude:
    - Network-level metadata (HTTP headers, etc.)
    - Non-structural timestamps (like arrival time)

unless explicitly included in the Deed's schema

canonicalization_profile:

unicode_normalization: NFC

encoding: UTF-8

mapping_key_order: lexicographic_recursive

scalar_normalization:

```
    booleans: lowercase
    null: null
  excluded_fields:
    - transport_metadata
  included_fields:
    - deed_id
    - event_type
    - payload
    - signatures
    - meta
cvid:
  digest_suite:
    default: BLAKE3
    allowed:
      - BLAKE3
      - SHA-256
  format: <digest_suite>:<hex_digest>
```

6. Validation Flow (Algorithm)

6.1 High-Level Flow

Client → /validate → Validator

→ Load Deed

→ Canonicalize event → CVID

→ Derive A,S,T (+ optional G)

→ Load relevant Journal history

→ Verify signatures

→ Execute Vault Logic

→ Write Journal entry

→ Optionally write Registry receipt

→ Sign and return decision

6.2 Validation Flow (Canonical YAML Steps)

```
validation_flow:
  - step: load_deed
    outputs: [deed]
  - step: canonicalize_event
    outputs: [canonical_bytes]
  - step: compute_cvid
```

```
  outputs: [event_cvid]
- step: derive_coordinates
  outputs: [A, S, T]
- step: load_history_slice
  selector: bounded_by_deed_policy
  outputs: [history]
- step: verify_signatures_and_roles
  outputs: [auth_ok]
- step: evaluate_vault_logic
  outputs: [decision, error_code]
- step: append_journal_entry
  required: true
  outputs: [journal_entry_id]
- step: append_registry_receipt
  condition: decision == ACCEPT and deed.registry_policy.publish_on_accept == true
  outputs: [registry_receipt_id]
- step: sign_decision
  outputs: [validator_signature]
- step: return_response
```

7. Vault Logic Execution

7.1 Requirements

- Non-Turing, declarative
- Guaranteed termination
- Deterministic (no randomness, no time-based branching)
- Access only to:
    - Canonical event fields
    - Derived coordinates
    - A well-defined slice of Journal history

- Static Deed configuration

7.2 Example Vault Logic (Predicate DAG in Canonical YAML)

```
vault_logic:
  model: predicate_dag
  nodes:
    - id: roles_ok
      predicate:
        kind: all_roles_present
        roles: [OpsEngineer, RiskOwner]
      on_fail:
        decision: REJECT
        error_code: E_AUTH
      next: env_ok
    - id: env_ok
      predicate:
        kind: in_set
        path: $.payload.environment
        set: [prod, staging]
```

- As an interpreter, or
- By compiling it into host-language predicates during Deed load

---

8. External API (Suggested)

8.1 HTTP Endpoint (Bodies in Canonical YAML)

POST /validate

Request body: Event object (as defined in Section 4.1)

Response: Decision object

Example:

POST /validate HTTP/1.1

deed_id: EX.AI.DeployGate.Model

event_type: MODEL_DEPLOY_REQUEST

payload:

  model_hash: abc123

  environment: prod

  requested_by: user-123

  ticket_id: IT-5621

  ts_client: 2025-10-10T13:45:00Z

signatures: ...

meta:

  request_id: ...

Response:

decision: ACCEPT

error_code: null

deed_id: EX.AI.DeployGate.Model

event_cvid: blake3:abcd...

journal_entry_id: JOR-2025-10-10-000001

registry_receipt_id: REG-2025-10-10-000001

ts_validator: 2025-10-10T13:46:09Z

validator_signature: base64...

---

9. Implementation Architecture (YAML-Native, No Host-Language Code)

This section specifies component boundaries and contracts without implementation-language examples.

validator_service:

components:

deed_store:

responsibility: load_versioned_deeds_from_yaml

journal_store:

responsibility: append_only_evidence_log_with_hash_chain

registry_store:

responsibility: minimal_receipts_append_only_with_integrity

crypto_provider:

responsibility: signature_verification_and_validator_signing

handler_contract:

input: canonical_event

output: signed_decision_object

constraints:

determinism: required

idempotency: required

side_effects: [append_journal, append_registry_if_allowed, emit_metrics]

10. Logging, Metrics & Testing

10.1 Logging

At minimum, log:

- deed_id, event_type
- decision, error_code
- Latency
- Any internal exceptions (without leaking sensitive payloads)

10.2 Metrics

Useful metrics:

- Number of validations per Deed
- ACCEPT vs REJECT counts
- Error code distribution (E-SIG, E-AUTH, etc.)
- Latency percentiles
- Journal/Registry append failures

10.3 Testing

Create test suites for:

- Canonicalization & CVID stability
- Signature failure scenarios (E-SIG)
- Scope violations (E-SCOPE)
- Turn-order failures (E-TURN)

- Duplicate / replay (E-DUPE)
- Chain tampering (E-CHAIN)

Each test should:

1. Prepare a Deed + Journal state
2. Call the validator
3. Assert decision + error_code
4. Assert Journal / Registry entries

---

11. Deployment Patterns & Security

11.1 Deployment

- Run validators as stateless services behind a load balancer where possible
- Journal and Registry stores should be:
    - Append-only
    - Tamper-evident (hash-linked)
    - Backed-up and replicated

11.2 Security Considerations

- Protect validator private keys (HSM or equivalent)
- Enforce mTLS between clients and validators for sensitive domains
- Restrict which Deeds a given validator instance may serve
- Ensure proper role separation:
    - Admins who deploy code ≠ operators who run validators ≠ key custodians

---

12. How to Use This Document

For an implementation partner (e.g. large Host-Entity SI or internal GovTech team):

1. Use Sections 1–4 to understand the role & data model
2. Use Sections 5–8 as the main engineering spec
3. Follow Section 9 as a starting architecture & code skeleton
4. Apply Sections 10–11 for non-functional requirements

END OF DOCUMENT 2

DOCUMENT 3

EGYPT AI DEPLOY GATE IMPLEMENTATION GUIDE v1.0

Document Type: Internal Technical Guide / Partner Handout
Scope: First pilot domain – AI Deploy Gate for a high-risk model (e.g. fraud detection, credit scoring, border/security)

---

1. Purpose

This guide describes how to implement an RSBIS Validator for Host Entity's AI Deploy Gate:

- It explains the end-to-end flow from AI deployment request → validator → Journal/Registry
- It defines a concrete Deed for the AI gate
- It gives example data structures, APIs, and pseudo-code
- It is intended for the Host-Entity implementation partner and/or internal government tech teams

---

2. Domain Overview: "AI Deploy Gate – a designated high-risk model"

Goal of this pilot domain

No high-risk AI model deployment can occur unless it passes:

- Structural identity checks (who/what is acting)
- Signature and role checks
- Turn-order and change-management checks
- Policy checks (only approved model hashes, environments, scopes)

The validator does not run the AI model itself; it approves or blocks deploys structurally.

---

3. Deed Definition (Host Entity AI Deploy Gate)

3.1 Deed ID

EX.AI.DeployGate.Model

- EG – Host Entity
- AI – Domain
- DeployGate – Function
- ModelX – Specific high-risk model governed by this Deed

3.2 Deed Structure (Example YAML)

deed_id: EX.AI.DeployGate.Model

version: 1

owner: "Host Entity – [Lead Institution]"

description: >

Structural gate for deployments of high-risk AI a designated high-risk model into production

and staging environments.

scope:

event_types:

```
    - MODEL_DEPLOY_REQUEST
  environments:
    - "prod"
    - "staging"

vault_logic:
  model: predicate_dag
  constraints:
    evaluation_graph: acyclic
    recursion: forbidden
    unbounded_iteration: forbidden
  nodes:
    - id: roles_ok
      predicate:
        kind: all_roles_present
        roles:
          - OpsEngineer
          - RiskOwner
      on_fail:
        decision: REJECT
        error_code: E_AUTH
      next:
        - env_ok
```

```
  - id: env_ok
    predicate:
      kind: in_set
      path: $.payload.environment
      set:
        - prod
        - staging
    on_fail:
      decision: REJECT


signing_policy:
  required_roles:
    - "OpsEngineer"
    - "RiskOwner"
  accepted_validator_keys:
    - "VAL-EX-AI-01"
    - "VAL-EX-AI-02"


registry_policy:
  publish_on_accept: true
  fields:
    - "event_type"
    - "payload.model_hash"
```

```
    - "payload.environment"
    - "payload.ticket_id"

crypto_policy:
  hash_algorithm: "BLAKE3"
  signature_algorithm: "Ed25519"

journal_policy:
  retention: "10y"
  encryption: "at_rest"
  access_control: "Host Entity-only, per-institution"

context_data:
  allowed_model_hashes:
    - "blake3:1234abcd..."  # model version 1
    - "blake3:5678ef90..."  # model version 2
```

---

4. Event & API Design

4.1 Incoming Event Shape

POST /validate

deed_id: EX.AI.DeployGate.Model

event_type: MODEL_DEPLOY_REQUEST

payload:

```
  model_hash: blake3:1234abcd...
  environment: prod
  requested_by: user-123
  ticket_id: IT-5621
  ts_client: 2025-10-10T13:45:00Z
  deploy_sequence: 17
signatures:
  - key_id: KEY-OPS-001
    role: OpsEngineer
    signature: base64-ops-signature
  - key_id: KEY-RISK-001
    role: RiskOwner
    signature: base64-risk-signature
meta:
  request_id: uuid-1234
  source_system: ci-cd-pipeline-1
```

4.2 Validator Response

decision: ACCEPT

error_code: null

deed_id: EX.AI.DeployGate.Model

event_cvid: blake3:abcd...

journal_entry_id: JOR-EX-AI-2025-10-10-000017

registry_receipt_id: REG-EX-AI-2025-10-10-000010

ts_validator: 2025-10-10T13:46:09Z

validator_id: VAL-EX-AI-01

validator_signature: base64-validator-sig

CI/CD or deployment system behavior:

- If decision = "ACCEPT" → continue deployment
- If decision = "REJECT" → block deployment and surface error_code + explanation

---

5. Canonicalization & CVID (Host Entity AI Gate)

Canonicalization rules (for this domain):

- Use UTF-8, Unicode NFC
- Remove non-structural network metadata
- • Canonical serialization: sorted keys, no pretty-print, no extra whitespace

- Include full payload, signatures, meta, and deed_id, event_type in canonical form

```
canonicalization_profile:
  reference: CanonicalSortSpec
  encoding: UTF-8
  unicode_normalization: NFC
  mapping_key_order: lexicographic_recursive
  scalar_normalization:
    booleans: lowercase
    null: null
  included_fields:
    - deed_id
    - event_type
    - payload
    - signatures
    - meta
  excluded_fields:
    - transport_metadata
cvid:
  digest_suite:
    default: BLAKE3
```

---

6. Structural Coordinates (A, S, T) – AI Gate

For this pilot, we recommend a simple rule:

- A (Ancestry) – identifies the "family" of this model's deployments
  e.g. A = 1 for ModelX in the pilot
- S (Stage) – environment stage
  - S = 0 → staging
  - S = 1 → prod
- T (Turn-Order) – strictly increasing deploy sequence for a given (A, S)

coordinates_policy:

A:

kind: constant

value: 1

S:

kind: enum_map

source_event_path: $.payload.environment

map:

staging: 0

prod: 1

T:

kind: event_field_integer

source_event_path: $.payload.deploy_sequence

monotonicity: strictly_increasing_per_(A,S)_over_ACCEPTs

Turn-order enforcement (Vault Logic + history):

- Load last ACCEPT for (A, S)
- Require T > last_T
- If not, return REJECT:E-TURN

---

7. Full Validation Flow (Host Entity AI Gate)

1. Receive request /validate
2. Load Deed EX.AI.DeployGate.Model
3. Canonicalize event → CVID
4. Derive (A, S, T)
5. Load history for (deed_id, A, S) from Journal
6. Verify signatures (Ops + Risk)
7. Execute Vault Logic (roles, model hash, ticket, turn-order)
8. Append Journal entry (always)
9. Append Registry receipt (on ACCEPT, per policy)
10. Sign decision and return

---

8. Minimal Implementation Checklist (for Partner)

- [ ] HTTP POST /validate endpoint
- [ ] Canonicalization + BLAKE3 hashing
- [ ] Coordinate derivation (A, S, T) as defined
- [ ] Journal store (append-only, hash-linked)
- [ ] Registry store (append-only, minimal receipts)
- [ ] Deed loader (YAML → in-memory config)

- [ ] Vault Logic interpreter for the AI Gate Deed
- [ ] Signature verification (Ed25519)
- [ ] Validator signing key & signatures
- [ ] Logging + metrics (per Deed, per error code)
- [ ] Unit tests for:
    - Valid deploy (ACCEPT)
    - Missing role/signature (E-SIG / E-AUTH)
    - Unapproved hash (E-MODEL)
    - Wrong environment (E-SCOPE)
    - Turn-order violation (E-TURN)

You can hand this guide directly to a partner and say:

"This is how to implement the AI Deploy Gate validator for the first pilot."

END OF DOCUMENT 3

DOCUMENT 4

VALIDATOR REQUIREMENTS FOR IMPLEMENTATION PARTNER

Document Type: Attachment for RFP / Partner Scoping
Audience: Systems integrators / large Host-Entity tech company

---

1. Project Context

The Host Entity is evaluating the RSBIS structural trust protocol for:

- AI governance

- Treasury/payments control
- Sectoral trust labels (e.g. tourism)

The first pilot domain is an AI Deploy Gate: a structural gate in front of a high-risk AI model that enforces:

- Approved model hashes
- Proper change-management
- Required sign-offs
- Verified, tamper-evident evidence

Root Zero Vault (RZV) is the protocol steward.
Host Entity will be the Deed holder.
The chosen implementation partner will build and operate the validator and integration layer.

---

2. Scope of Work – Validator & Integration

The partner is expected to deliver a production-grade validator service for one pilot Deed (AI Deploy Gate), including:

2.1 Validator Service

Build a deterministic validator that:

- Exposes an HTTP API, e.g. POST /validate
- • Accepts structured events (canonical YAML mapping) representing AI deployment requests
- Loads and applies Deed configuration (YAML)
- Canonicalizes events and computes a CVID (BLAKE3)
- Derives structural coordinates (A, S, T) as defined by the Deed
- Loads relevant Journal history for context
- Verifies signatures according to the Deed's signing policy
- Executes the Deed's Vault Logic (non-Turing, deterministic rules)

- Writes an append-only Journal entry for every validation
- Optionally writes a Registry receipt for accepted events
- Returns a signed decision object:

decision: ACCEPT

error_code: null

deed_id: ...

event_cvid: ...

journal_entry_id: ...

registry_receipt_id: null

ts_validator: ISO8601

validator_id: VAL-...

validator_signature: signature...

2.2 Deed & Vault Logic Integration

The partner will:

- Implement a Deed loader (YAML → in-memory config)
- Implement a Vault Logic interpreter or simple rule engine sufficient for:
    - Role / signature checks
    - Membership checks (e.g. model hash in approved list)
    - Simple field predicates (non-null, enumerations)
    - Turn-order enforcement using Journal history
- Support hot-reload or controlled rollout of updated Deeds (v1, v2, etc.)

RZV will provide:

- Example Deed for the AI Gate
- Vault Logic DSL specification
- Reference pseudo-code and review

2.3 Journal & Registry Storage

The partner will design and implement:

A Journal store:

- Append-only
- Hash-linked entries (to detect tampering)
- Reasonable indexing for pilot (by deed_id, (A, S), timestamps)
- At-rest encryption and access controls

A Registry store:

- Append-only
- Minimal receipts only (fields specified by the Deed)
- Public or inter-institutional API (if requested by Host Entity)

Preferred technologies (for pilot):

- Postgres / similar relational DB plus a hash-chain design, or
- Another Host Entity-approved database with an explicit hash-chain layer

2.4 Integration with AI Deployment Pipeline

The partner will:

- Integrate the validator with the existing CI/CD or MLOps pipeline for the selected AI model:
    - A deployment request must call /validate before proceeding
    - If the validator returns REJECT, the pipeline must abort the deploy and surface the reason to operators

- Work with the AI/DevOps team to:
    - Define the exact fields in payload (model hash, environment, ticket id, sequence number, etc.)
    - Ensure that the model hash is computed and available at validation time

---

3. Non-Functional Requirements

3.1 Determinism & Idempotence

- Same input (identical canonical event) must always produce:
    - Same decision
    - Same error_code
    - Same event_cvid
- Re-validating the same event (same event_cvid) must not create conflicting or duplicate Journal entries

3.2 Security

- All validator decisions must be signed with a validator key stored securely (HSM or equivalent)
- All inbound calls to the validator must:
    - Use TLS
    - Optionally enforce mutual TLS (mTLS) for internal systems
- The partner must implement:
    - Role-based access control for Journal/Registry access
    - Secure storage for keys and configuration

3.3 Logging & Metrics

Minimum metrics:

- Validations per Deed per time unit
- ACCEPT vs REJECT counts

- Error code frequencies (E-SIG, E-AUTH, E-TURN, etc.)
- Latency distribution (p50, p95)
- Journal/Registry write failures

Minimum logs:

- deed_id, event_type, decision, error_code, ts_validator, request_id
- (no unnecessary sensitive payload in logs)

3.4 Performance & Availability (for pilot)

Target throughput for pilot:

- Low volume (e.g. tens to hundreds of validations/day) – high correctness, not high TPS

Availability:

- Validator should be designed for HA (e.g. redundant instances behind load balancer)
- In case of validator outage, the default should be fail-closed for high-risk deploys (no deploy without a structural decision), unless Host Entity explicitly decides otherwise

---

4. Deliverables

The partner will deliver:

1. Validator Service v1
    - Running on Host-Entity infrastructure
    - Serving the AI Deploy Gate Deed
2. Journal & Registry Stores
    - Schema and implementation
    - Hash-chain logic
3. Integration Hook
    - Working integration into the chosen AI deployment process

- Demonstrable: a deploy is blocked if validator returns REJECT

4. Configuration & Deeds
    - Deed configuration as agreed with Host Entity and RZV
    - Vault Logic implementation for the AI Gate
5. Documentation
    - Technical run-book (deployment, keys, monitoring, backup)
    - API reference for /validate
    - Admin / operator guide
6. Test Suite
    - Automated tests covering:
        - Valid deploy (ACCEPT)
        - Signature failures
        - Unapproved model hash
        - Wrong environment
        - Turn-order violation
        - Tampered Journal / detection

---

5. Skills & Experience Expected

The implementation partner should demonstrate capabilities in:

- Secure backend development (any modern language; partner choice)
- Cryptography integration (hashing, digital signatures)
- API design and integration with CI/CD / MLOps
- Database design for append-only, audit-grade logs
- DevOps / infrastructure on Host-Entity-approved platforms

- Working with government or regulated entities (preferred)

RZV will:

- Provide technical specs and reference designs
- Participate in architecture reviews
- Review implementations for structural correctness and thin-law alignment

---

END OF DOCUMENT 4

DOCUMENT 5

VALIDATOR OPERATIONS RUNBOOK v1.0

Use: Internal doc for Host Entity + implementation partner operations teams.

---

1. Purpose

This runbook explains how to operate the RSBIS Validator in production for the pilot:

- How it is deployed and configured
- How to change rules safely (Deed updates)
- How to handle incidents
- How to monitor, backup, and audit the system

---

2. Architecture Snapshot

Components in the pilot:

1. Validator Service

- Stateless or light-state HTTP service
- Runs the Deed + Vault Logic on incoming events
- Signs decisions with a validator key

2. Journal Store
    - Append-only log with hash-chained entries
    - Private, encrypted, access-controlled
3. Registry Store
    - Append-only receipts (subset of fields)
    - Exposed via internal read API (and optionally public views)
4. Configuration & Deeds
    - Deeds stored as versioned YAML
    - Loaded on startup or via controlled reload
5. Monitoring & Logging
    - Metrics (Prometheus/Grafana or equivalent)
    - Structured logs to SIEM / log platform
6. Integrations
    - CI/CD or AI deployment system calling /validate

---

3. Environments

Recommended minimum:

- DEV – for engineers validating changes
- STAGING – mirrors production configuration, testing integration end-to-end
- PROD – live validations for the pilot domain

Rule: Changes must move DEV → STAGING → PROD with approvals.

4. Configuration & Deed Management

4.1 Deed Storage

Deeds stored in a versioned repository (Git or equivalent).

Directory example:

deeds/

EX.AI.DeployGate.Model/

v1.yaml

v2.yaml

EG.TOUR.TrustLabel/

v1.yaml

4.2 Change Process (Deed/Vault Logic)

1. Propose change
    - Governance / policy team updates YAML (e.g. new allowed_model_hashes)
2. Review
    - Technical lead + governance representative review and approve
3. Test in DEV
    - Deploy updated Deed to DEV Validator
    - Run automated tests (happy path + all error codes)
4. Promote to STAGING
    - Repeat tests with real integration
5. Promote to PROD
    - Scheduled maintenance window (if needed)

- Apply config change and verify metrics

All Deed changes should be:

- Tracked (Git commit history)
- Tagged with change ticket ID
- Documented in a short change log

---

5. Deployment & Rollout

5.1 Initial Deployment Steps

1. Provision infrastructure (VMs/containers) for Validator
2. Provision databases for Journal & Registry
3. Generate validator key pair and store private key in a safe location (e.g. HSM or secrets manager)
4. Configure:
   - Deed(s)
   - Database connection
   - Logging and metrics
   - Allowed calling systems (IP allowlist / mTLS)
5. Deploy Validator in STAGING
6. Integrate with CI/CD in STAGING
7. Run end-to-end tests
8. Repeat for PROD

5.2 Rollback Strategy

If a new Validator version or Deed causes issues:

Option A – Config rollback
Revert to previous Deed/logic from Git and redeploy

Option B – Binary rollback
Redeploy previous Validator container/image

Because Journal/Registry are append-only and versioned, rollbacks do not corrupt history; they only affect new decisions.

---

6. Key Management

6.1 Validator Keys

- Generate validator key pair (e.g. Ed25519)
- Store private key in:
    - HSM, or
    - Cloud KMS, or
    - Dedicated secure vault
- Configure Validator to sign decisions with this key
- Publish public key to:
    - Host Entity's internal trust store
    - RZV (if needed for global verification)

6.2 Key Rotation

1. Add new key to Deed's accepted_validator_keys with valid_from date
2. Deploy Validator using the new key
3. After overlap period, remove old key from Deed
4. Journal and Registry entries remain verifiable using the correct historical key

Record all key changes in a key rotation log.

---

7. Monitoring, Alerts, and Health

7.1 Metrics to Track

- Request volume
  validator_requests_total (labels: deed_id, decision)
- Error distribution
  validator_errors_total (labels: error_code)
- Latency
  validator_request_duration_seconds (p50, p95)
- Journal/Registry writes
  journal_write_failures_total
  registry_write_failures_total

7.2 Suggested Alerts

Trigger alerts when:

- Validator returns REJECT for > X% of requests in a 5–15 min window
- journal_write_failures_total > 0
- registry_write_failures_total > 0
- Latency spikes beyond defined threshold
- Validator instances become unavailable

Rule of thumb for pilot:

Any failure to write to Journal = SEV-1
(Because it jeopardizes evidence.)

---

8. Backup & Recovery

8.1 Journal Backup

- Daily full backup, plus incremental if needed
- Store backups:
  - Encrypted

- In at least two physically separate locations

8.2 Registry Backup

- Similar schedule; may be less sensitive but still important for continuity

8.3 Recovery Procedure

1. Restore database from most recent known-good backup
2. Verify hash chain integrity:
    - For Journal: each entry's prev_hash must match previous entry's hash
    - Investigate any break in the chain
3. Once integrity confirmed, re-enable Validator writes

Document RTO (Recovery Time Objective) and RPO (Recovery Point Objective) for pilot.

---

9. Incident Response

9.1 Types of Incidents

- Validator outage (service not reachable)
- Write failure (cannot write to Journal/Registry)
- Data integrity issue (hash chain break)
- Unauthorised access or attempted tampering

9.2 Standard Response Steps

1. Detect & Triage
    - Alert triggers
    - On-call engineer acknowledges
2. Stabilize
    - For high-risk flows, default is fail-closed:
        - No deploy without successful validation

    - Communicate impact to relevant teams (AI, ops, governance)
3. Investigate
    - Check logs, metrics, DB status
    - Verify integrity of hash chains
4. Remediate
    - Fix the cause (config, infra, storage)
    - Restore from backup if needed
    - Rotate keys if compromise suspected
5. Document
    - Incident report: timeline, root cause, fix, lessons

---

10. Routine Operational Tasks

Daily / Weekly / Monthly tasks:

Daily

- Check dashboard (errors, latency, unusual spikes)
- Confirm backups completed successfully

Weekly

- Spot-check Journal & Registry integrity
- Review any REJECT spikes or new error patterns

Monthly

- Review Deeds & Vault Logic with governance team
- Review key rotation plan and upcoming expiries
- Test disaster recovery procedure (at least quarterly)

END OF DOCUMENT 5

DOCUMENT 6

SECURITY & COMPLIANCE ANNEX v1.0

RSBIS / Root Zero Vault – Security & Compliance Annex
Validator, Journal, Registry

Draft v1.0 – For Pilot & Partner Discussion

---

1. Purpose

This annex defines the security, integrity, and compliance requirements for any implementation of the:

- RSBIS Validator (decision engine)
- Journal (private canonical evidence log)
- Registry (minimal public receipts)

Written for:

- Government security, risk, and compliance teams
- Implementation partners and system integrators
- Internal IT / DevOps / SRE teams
- External auditors and regulators

It complements:

- Concept Note & Founding Offer
- Pilot Implementation Blueprint
- Validator Overview & Implementation Guide
- Operations Runbook

2. Threat Model

RSBIS does not store citizen data or run AI models. Its critical role is to decide which high-risk actions are allowed to happen, and to anchor evidence of those decisions.

2.1 Primary Threats

1. Policy Subversion
    - Unauthorized changes to Deeds or Vault Logic
    - Effect: malicious or overly-permissive rules in PROD
2. Validator Compromise
    - Attacker controls validator binary or process
    - Effect: forged ACCEPT/REJECT decisions
3. Journal Tampering / Loss
    - Modifying, deleting, or reordering Journal entries
    - Effect: evidence cannot be trusted
4. Registry Poisoning
    - False receipts injected into Registry
    - Effect: external verifiers trust incorrect claims
5. Key Theft / Misuse
    - Theft of validator or Deed-signing keys
    - Effect: valid-looking but malicious signatures
6. Denial of Service (DoS)
    - Validator becomes unavailable
    - Effect: high-risk actions cannot be validated (correctly fail-closed, but operationally disruptive)

2.2 Structural Mitigations (By Design)

RSBIS is built so that security is architectural, not bolted on:

Thin-Law Deeds
Immutable baseline rules; Deed changes are treated as constitutional amendments, not config tweaks

Canonical CVID + Hash Chains
Events and entries are canonicalized and hashed; Journal entries are hash-chained; tampering becomes detectible

Fail-Closed for High-Risk Flows
If the validator or Journal is unavailable, high-risk actions do not proceed

Separation of Concerns
Deed (policy) → Validator (decision) → System (execution) are separate components with separate access controls

---

3. Identity & Access Management (IAM)

3.1 Roles

At minimum, the following roles must be defined and enforced:

- Deed Author – drafts Deed YAML / Vault Logic; no PROD deploy rights
- Deed Approver – governance/legal/owner who approves Deeds; does not operate infrastructure
- Validator Operator – deploys and monitors validator; cannot change Deeds or keys
- Security Admin – manages keys, HSMs, and infra security; cannot change business policy
- Auditor – read-only access to logs and reports; no write access anywhere

3.2 Principles

Least Privilege
Each role has only the permissions required

Separation of Duties
No single person can:

- Modify Deeds and deploy them to PROD, or

- Change validator code and control validator keys

Strong Authentication

- MFA required for all privileged actions
- Privileged Access Management (PAM) recommended

---

4. Key Management

Validator and Deed-signing keys must be treated as critical national assets.

4.1 Storage & Generation

- Keys generated and stored in:
  - HSMs or certified secure key management systems
- No private keys stored:
  - In source code
  - On developer laptops
  - In plain-text configuration files

4.2 Lifecycle & Rotation

Each Deed defines allowed validator keys:

```
accepted_validator_keys:
  - key_id: "VAL-EX-AI-01"
    valid_from: "2025-01-01"
    valid_until: "2025-12-31"
  - key_id: "VAL-EX-AI-02"
    valid_from: "2025-12-01"  # overlap for rotation
    valid_until: null
```

Requirements:

- Planned rotation (e.g., annually) plus emergency rotation
- Overlap windows for smooth migration
- Revocation procedure for compromised keys
- Full logging of key creation, rotation, and revocation

4.3 Usage

- All validator responses are signed with the active validator key
- Journal entries may be signed or MAC'd; integrity is always ensured via hashing + hash chain

---

5. Network & Infrastructure Security

5.1 Segmentation

- Validator runs in a secured network segment:
    - Only authorized systems (e.g., CI/CD pipeline, treasury system) can call /validate
- No public internet exposure unless explicitly required and carefully controlled (e.g., through an API gateway)

5.2 Encryption

- In Transit: All API traffic uses TLS with mutual authentication (mTLS)
- At Rest: Journal and Registry storage encrypted using strong algorithms (e.g., AES-256)

5.3 Hardening

- OS and runtime hardened using recognized benchmarks (e.g., CIS)
- No direct root/admin logins to PROD; all access via controlled mechanisms with logging
- Regular patching and vulnerability remediation for OS, runtime, and dependencies

---

6. Application Security

6.1 Deterministic Behavior

- The Vault Logic engine must be non-Turing complete and deterministic:

same Deed + same event → same decision every time

- No random sources, time-dependent branching, or external mutable state can influence core decision logic

6.2 Input Validation

- Strict schema validation of validator requests:
    - Allowed event_types only
    - Required fields present and correctly typed
    - Signature objects must be structurally valid
- Malformed requests return REJECT with specific error codes (e.g., E-FORMAT)

6.3 Code Quality & Review

- Mandatory peer review for:
    - Validator code
    - Deed parsing logic
    - Vault Logic interpreter
- Static Analysis (SAST) integrated into CI/CD
- Security-focused code review for cryptographic operations and hashing

6.4 Supply Chain Security

Dependencies:

- Pinned to exact versions
- Scanned for known vulnerabilities (SCA)
- Fetched from trusted repositories or internal artifact stores

Maintain a Software Bill of Materials (SBOM) for the validator component

7. Evidence, Audit, and Integrity

7.1 Journal Requirements

Each Journal entry MUST contain at least:

- journal_entry_id
- deed_id
- event_cvid
- decision (ACCEPT/REJECT)
- coords (A, S, T)
- prev_journal_hash
- hash (current entry hash)
- ts_validator
- validator_signature
- Optionally, event_snapshot and additional metadata

Hash Chain Integrity:

- Each entry's prev_journal_hash must equal the previous entry's hash
- Any break indicates tampering or data loss and triggers an incident

Backups:

- Encrypted backups taken regularly (e.g. daily)
- Periodic restore tests to ensure recoverability
- Retention according to national policy (e.g. 7–10 years for critical domains)

7.2 Registry Requirements

- Stores minimal, non-sensitive receipts:
    - E.g., event type, timestamp, Deed ID, event CVID, high-level summary fields

- Entries are integrity-protected and auditable
- Provides APIs for:
    - Query by Deed
    - Query by date range
    - Query by event CVID / model hash / transaction ID, etc.

7.3 Administrative Logging

All administrative actions must be logged:

- Deed creation / modification / deployment
- Key operations (generate / rotate / revoke)
- Validator configuration/deployment changes

Logs:

- Sent to centralized SIEM
- Time-synchronized
- Protected from tampering (e.g. WORM storage or log hashing)

---

8. Disaster Recovery & Incident Response

8.1 Recovery Objectives

RPO (Recovery Point Objective):
Aim: zero loss of Journal; practically, accept only documented gaps between backup and incident

RTO (Recovery Time Objective):
Defined per domain; high-risk systems should have tighter RTO

8.2 Standard Incident Scenarios

1. Validator Outage

Effect: high-risk actions blocked (fail-closed)

Response:

- Diagnose infra/network/app cause
- Restore service
- Verify Journal and Registry integrity
- Document downtime and affected requests

2. Journal Corruption or Loss

Effect: evidence gaps or broken hash chain

Response:

- Restore from last good backup
- Verify hash chain
- Document gap and assess whether any decisions in the gap must be re-validated or re-enacted

3. Key Compromise

Effect: untrusted validator decisions in exposure window

Response:

- Immediately revoke compromised key in relevant Deeds
- Rotate to new key(s); update accepted_validator_keys
- Audit all decisions in suspected window
- Consider compensating controls or re-validation of impacted events

4. Deed Tampering

Effect: unauthorized or malicious policy in effect

Response:

- Compare PROD Deed to version-controlled history
- Revert to last approved version
- Audit decisions under tampered Deed

- Tighten Deed change approvals and monitoring

---

9. Compliance Alignment (High-Level)

This section is descriptive, not legal advice. It explains how RSBIS features support typical legal and regulatory obligations.

9.1 Governance & Accountability

Laws and regulators increasingly require:

- Clear who/what/when/why for high-risk decisions

RSBIS provides:

- Explicit Deeds (policy)
- Signed validator decisions
- Hash-chained Journal entries
- Bounded, offline re-computation of decisions

9.2 Integrity & Non-Repudiation

Traditional logs can be edited without detection.

RSBIS:

- Uses CVID (canonical hash) and Journal hash chains
- Signs decisions with validator keys
- Stores Registry receipts for external verification

9.3 Data Minimization & Sovereignty

- Validator, Journal, and Registry can be fully deployed in national data centers
- Registry receipts are minimal and non-sensitive
- No requirement to export citizen data or operational logs to Root Zero Vault or foreign infrastructure

9.4 AI Governance & Risk

Emerging AI frameworks (e.g., EU AI Act, national AI strategies) demand:

- Traceability of model versions
- Explicit approvals
- Controls on high-risk deployments

RSBIS AI Deploy Gate:

- Enforces whitelisted model hashes
- Enforces required sign-off roles
- Records all deploy attempts and outcomes

---

10. Security & Compliance Checklists

10.1 Before Pilot Go-Live

- [ ] Roles & RBAC defined (Author, Approver, Operator, Security, Auditor)
- [ ] Validator & Journal infra deployed in segmented networks
- [ ] TLS/mTLS configured and tested
- [ ] Keys generated in HSM/secure store; no plaintext keys
- [ ] Deed governance workflow documented and approved
- [ ] Monitoring, metrics, and alerting configured
- [ ] Backup and restore tested at least once
- [ ] Initial penetration test / security review completed; critical issues resolved

10.2 During Pilot

- [ ] All Deed changes reviewed, approved, and logged
- [ ] All key operations logged and reviewed
- [ ] Journal backups running and restore tests scheduled

- [ ] Security alerts monitored; incidents documented

10.3 After Pilot

- [ ] Security posture review: strengths, weaknesses, lessons
- [ ] Compliance review: auditability, reporting, regulator feedback
- [ ] Recommendations for scaling to additional Doors and validators

---

11. Summary

This annex defines how the RSBIS Validator, Journal, and Registry must be:

- Secured (keys, infra, IAM, deterministic behavior)
- Auditable (hash-chains, logs, signatures)
- Compliant-ready (governance, AI-risk, data sovereignty)

It is intentionally generic enough to fit Host Entity's own security standards and regulations, while being specific enough for partners to implement and for auditors to evaluate.

END OF DOCUMENT 6