Raziel

Agent Authorisation at Superhuman Scale

Final Project Report

Gita Alekhya Paul

Masters in Digital Financial Technology
FT5007 Capstone Project

Advisor: Dr. Jungpil Hahn

National University of Singapore

30 October 2026

Executive Summary

Organizations need agents to use data without granting blanket access. Trust must scale with automated data access. Google's Zanzibar serves millions of authorization requests each second [20]. Human choices face limits of information and computation [29]. An automated caller can repeatedly test a permissions boundary. This leads us to a practical question: how can agents explore useful data without receiving the private relationships behind access?

We developed Raziel to support exploration within a granted task scope. The authorization layer combines SpiceDB with a local SP1 Plonk prover. A compiler translates supported policies into PolicyIR. Canonical commitments bind the policy and a blinded relationship snapshot. An owner-side service constructs the private witness. A separate verifier checks the proof against its trusted policy registry. This addresses approved policies and authorization before execution in *Safeguards for Agentic Finance at Runtime* (SAFR) [28] paper by Monetary Authority of Singapore.

Each capability binds an agent key and task scope. Every protected request needs a fresh signature and nonce. Verified claims support repeated queries without another proof. Private relationships and witness material remain with the owner. The verifier receives the approved policy and public proof.

We demonstrated two-agent payment reconciliation through an operator console, the public `@gitaalekhyapaul/raziel-sdk-ts` package, and separate agent dashboards. Agent A reads processor records. Agent B uses its own capability for bank records. The demonstration returns an SGD 125.00 match with zero difference over labeled synthetic data.

The experiments show the cost of separating proof generation from verification. The eight-CPU run produced a 2,537-byte serialized proof artifact in 929.660 seconds, compared with 1,742.106 seconds in the four-CPU run. A retained real proof verified in 447 milliseconds inside a networkless worker with the issuer and prover stopped. The full service-flow experiment took 955.516 seconds to prove, peaked at 43.77 GiB, used no swap, and recorded no out-of-memory event. Negative tests rejected altered claims, wrong bindings, replay, and out-of-scope access.

We see Raziel as a foundation for private, verifiable agent-data access. Further research extends this model toward private denial proofs. Our longer-term vision for Hypha combines authorization with data discovery, relevance, and disclosure controls.

Contents

Executive Summary **i**

1 Business Context and Problem Statement **1**
1.1 Practical Problem and Motivation . . . 1
1.2 Stakeholders and Constraints . . . 2
1.3 Objectives, Scope, and Deliverables . . . 2
1.4 Report Organization . . . 3
1.5 Use of AI Tools and Declaration . . . 3

2 Requirements and User Stories **4**
2.1 Functional Requirements . . . 4
2.2 Analytic Requirements . . . 5
2.3 User Stories and Acceptance Criteria . . . 5
2.4 Non-Functional Requirements . . . 5

3 Literature Review and Methodology **7**
3.1 Literature Informing the Design . . . 7
3.2 Relevant Literature and Existing Systems . . . 7
3.2.1 Encrypted Search and Whole-Workflow Disclosure . . . 7
3.2.2 Cryptographic Access and Computation on Encrypted Data . . . 8
3.2.3 Private Attribute-Based Authorization . . . 8
3.2.4 Relationship Policies and Graph Enforcement . . . 9
3.2.5 Proof-Carrying Evidence and Reusable Authority . . . 9
3.2.6 Verifiable Execution and Privacy . . . 10
3.3 Project Development and Evaluation . . . 10
3.4 Evaluation Measures . . . 11

4 System Design and Protocol **12**
4.1 Authorization Statement and System Boundary . . . 12
4.2 Architecture and Trust Zones . . . 13
4.3 From a Visual Policy to Approved PolicyIR . . . 14
4.4 Relationship State and Snapshot-Time Authorization . . . 14
4.5 Asynchronous Issuance and Agent Consent . . . 15
4.6 Protected Requests and Safe Reuse . . . 16

5 Implementation and User Interface **17**
5.1 Repository and Language Boundaries . . . 17
5.2 Operator Interface . . . 17
5.3 Public SDK and External Agent Consumption . . . 18
5.4 Agent Tools and the LLM Boundary . . . 19
5.5 Container Deployment and Operational Controls . . . 21

6 Two-Agent Payment Reconciliation **22**
6.1 Business Task and Synthetic Ledgers . . . 22
6.2 Operator-Supervised Agent Workflow . . . 22
6.3 Authorization Controls in the Scenario . . . 23
6.4 Reconciliation Outcome and Capability Reuse . . . 23

7 Evaluation and Findings **27**
7.1 Evaluation Design . . . 27
7.2 Proof Generation and Resource Results . . . 27
7.3 Independent Verification and Mutation Results . . . 28
7.4 Capability Reuse and Cost Interpretation . . . 28
7.5 Answers to the Research Questions . . . 29
7.6 Operational Implications . . . 29

8 Conclusions and Recommendations **30**
8.1 Project Outcomes . . . 30
8.2 Future Work . . . 30
8.2.1 Freshness and Revocation as a Measured Tradeoff . . . 30
8.2.2 Comparative Performance and Larger Authorization Graphs . . . 31
8.2.3 From Positive Proofs to Private Denial . . . 31
8.2.4 Data Provenance, Relevance, and Cumulative Disclosure . . . 31
8.3 Raziel Within the Hypha Vision . . . 31

References **32**

A Reproducibility and Evidence Index **35**
A.1 Implementation Revisions . . . 35
A.2 Experiment Records . . . 35
A.3 Supplementary Workflow Artifact . . . 36

Chapter 1

Business Context and Problem Statement

1.1 Practical Problem and Motivation

Trust in delegated data access must hold across many decisions. Pang et al. report millions of authorization requests per second in Google's Zanzibar deployment [20]. At this scale, trust must be enforced through repeatable checks rather than personal familiarity with each caller.

We distinguish the universe of possible actions, the actions a role permits, and the actions a person considers taking. A person's perceived options can be narrower than their formal authority. Simon's account of bounded rationality connects choice to limited information and computational capacity [29]. Our concern is the gap between a formal permission and the willingness to explore it.

We cannot treat human hesitation as a security boundary for automated callers. An agent can vary requests and build up observations about which ones succeed. A compromised agent can direct that exploration toward an attacker's goal. AgentDojo studies this risk of redirected tool use through 97 tasks and 629 security test cases, including banking workflows [8].

This leads us to place authority in a task-scoped capability rather than the caller's willingness to stop probing. We want agents to explore useful data within an explicit grant without receiving the relationships that justify it. The design keeps authorization evidence checkable and the relationship witness with its owner.

Payment reconciliation gives this question a practical setting. We chose two agents that need limited access to separate ledgers. Raziel joins policy approval, committed relationship state, local Plonk proofs, and independent verification. Each capability proves `ALLOW` for its agent-bound scope [24].

We relate this design to *Safeguards for Agentic Finance at Runtime* (SAFR). Its Controls Repository holds applicable rules, and its Disposition Engine checks proposed actions [28, pp. 8, 13]. We focus on approved policy records and independent authorization before data release. Proof-backed capabilities support repeated signed queries, with receipts recording

their outcomes.

1.2 Stakeholders and Constraints

The data owner controls resources. Policy and snapshot authorities approve rules and authenticate relationship state. Operators prepare datasets for consumption and review requests from agents. In this proof-of-concept, agents use separate signing keys and scoped SDK requests. The gateway and independent verifier enforce access.

The verifier receives approved PolicyIR through its trusted registry. Agents receive the policy hash and public scope. Relationship tuples, satisfying paths, caveat values, salts, and Merkle openings remain inside the owner's proving boundary [24].

We ran proof experiments on the NUS virtual machine described in Chapter 7. Asynchronous jobs separate proof creation from protected requests. The relationship root identifies authorization state. The gateway separately binds policy and scope to a dataset revision and checksum.

1.3 Objectives, Scope, and Deliverables

We evaluate proof-carrying authorization under explicit policy, privacy, and resource bounds. Three questions guide the implementation and evaluation:

RQ1.
Can a finite supported SpiceDB fragment retain its authorization meaning through compilation into PolicyIR and produce real local Plonk proofs within measured resource constraints?

RQ2.
Can an independent verifier check the approved policy and snapshot-bound authorization statement without receiving the private relationship witness or contacting the issuer?

RQ3.
Can the deployed gateway reuse verified capability claims for bounded queries without removing agent signatures, scope checks, expiry checks, or replay protection?

We implemented the policy compiler, commitment and witness code, prover, independent verifier, gateway, SDK, and three dashboards. Semantic comparisons, negative tests, resource measurements, and the synthetic-data demonstration evaluate this system [24].

Raziel's current `ALLOW`-only scope excludes future `DENY` proofs and Hypha's broader governance. It does not deal with the broader implications of an agent identity and delegated authentication system. Proofs establish neither transaction truth nor reconciliation accuracy. Real-data acceptance, dataset relevance through secure zero-knowledge metadata queries, cumulative privacy accounting, and verifier-private policy remain separate research topics and serve as extensions to the base idea, eventually leading up to the creation of fully federated data networks through Hypha.

1.4 Report Organization

The following chapters cover requirements, literature, methods, design, implementation, demonstrations, evaluation, and research extensions.

1.5 Use of AI Tools and Declaration

We used ChatGPT and Codex to generate code from specifications written by us. We used these tools to prepare experiment reports from runs on the NUS VM cluster. We used these tools to prepare user acceptance records from stories tracked through the baton protocol. We used AI assistance to prepare this report from project work receipts, interim reports, video demonstrations, and feedback from our academic advisor. We reviewed and edited the generated material. We are responsible for the content and quality of the submitted work.

Chapter 2

Requirements and User Stories

2.1 Functional Requirements

The requirements connect policy authority, proof computation, and gateway enforcement to observable outcomes. [24]

F1: Compile and approve a bounded policy. Produce deterministic PolicyIR from supported SpiceDB expressions and reject unsupported constructs. A separate authority approves the canonical policy. The issuer cannot approve its own candidate. The verifier resolves its hash through a trusted local registry.

F2: Bind authorization to relationship state. Build each witness from one signed, committed snapshot revision. Role changes affect fresh snapshots, not the meaning of existing capabilities.

F3: Accept a signed, scoped request. Bind the agent key, task, purpose, action, resource scope, nonce, and validity. Expose asynchronous proof-job state without private inputs.

F4: Issue a proof-backed capability. Construct a bounded positive witness. The guest checks commitments, membership, supported caveats, and request binding. Validate the local Plonk result before publication.

F5: Verify independently. Check the proof and public claims against trusted policy and snapshot records without contacting the issuer. Issuer signatures cannot replace proof verification.

F6: Enforce bounded, reusable access. Check capability scope, validity, agent signature, and fresh request nonce on every query, including cache hits. Apply the immutable trusted dataset-revision mapping on both cold and cached paths.

F7: Reject invalid access without exposing witnesses. Return no protected rows for invalid signatures, replay, expiry, alteration, or scope mismatch. Keep agent-facing failures generic.

F8: Retain public evidence and dataset identity. Receipts retain safe capability, request, task, policy, and snapshot identifiers. Dataset records retain synthetic provenance, checksum, schema mapping, and row count. Relationship commitments do not commit transaction contents.

2.2 Analytic Requirements

Evaluation tests authorization, evidence integrity, disclosure boundaries, and cost, not payment accuracy. Controlled fixtures and recorded live runs supply distinct evidence. [24]

A1: Compare semantic decisions. For RQ1, compare supported SpiceDB and Rust cases against expected outcomes. Test rejected constructs separately. Agreement covers the tested fragment.

A2: Test proof and binding integrity. For RQ2, verify retained Plonk artifacts with issuer and prover stopped. Mutate proofs, keys, claims, and signed fields. Inspect public outputs for private witness disclosure.

A3: Evaluate enforcement across capability reuse. For RQ3, compare first and cached queries using distinct signed requests. Test scope, expiry, alteration, missing signatures, and replay on the enforcement path.

A4: Separate computational costs. Record execution, proving, proof size, verification, peak memory, and gateway costs separately. Name resource limits and fixture identity. Report feasibility, not production capacity.

2.3 User Stories and Acceptance Criteria

Table 2.1 links stakeholder intent to acceptance evidence. Local role checks and live authorization journeys retain separate evidence status. [24]

2.4 Non-Functional Requirements

Keep approved PolicyIR verifier-visible and witnesses owner-private. Use local proving, versioned canonical encodings, strict decoding, bounded collections, and bounded traversal. Public responses and logs exclude raw witnesses. Pinned tools and recorded conditions support reproduction.

Table 2.1: User stories, requirements, and acceptance evidence

Story	Actor, intent, and benefit	Links	Acceptance evidence
US-01	Policy owner: approve supported rules for accountable issuance.	F1, A1	SpiceDB agreement, rejected constructs, approved registry record.
US-02	Operator: manage roles for fresh access decisions.	F2, A1	Local membership checks and revision-pinned snapshot preparation.
US-03	Agent: request task-scoped authority without shared credentials.	F3, F4, A2	Signed request, retained Plonk capability, mutation rejection.
US-04	Gateway operator: verify without trusting an issuer's Boolean answer.	F5, A2	Verification after issuer shutdown. Proof and key mutations fail.
US-05	Agent: reuse authority without repeating proof generation.	F6, F7, A3, A4	Live cold and cached queries. Invalid requests return no rows.
US-06	Evaluator: trace access without private graph disclosure.	F8, A2, A4	Safe receipts, dataset provenance, checksum, recorded run conditions.

Keep agent keys under separate custody. Operators can inspect proof-job state without waiting for proving in a browser request. Snapshot validity, registry status, and configured revocation rules govern reuse. A later relationship edit does not itself invalidate the mathematical proof.

Authorities establish policy approval and snapshot provenance. The gateway enforces dataset mapping and request checks. These assumptions remain distinct from the ALLOW-only cryptographic statement. Rejection does not prove `DENY` in this version of implementation. [24]

Chapter 3

Literature Review and Methodology

3.1 Literature Informing the Design

Our early research asked how an agent could find useful information without exposing a private data collection. We examined searchable encryption, attribute-based encryption, and computation on encrypted data. These sources separated several questions that initially appeared to be one problem: finding matching records, controlling access, and limiting what repeated interactions reveal.

We then focused on the evidence required for an agent's access request. Private ABAC showed a route to authorization over hidden inputs. Zanzibar and SpiceDB supplied a relationship-based policy model. Proof-carrying authorization and verifiable computation supplied the basis for independent checking. We brought these ideas together in Raziel: an approved policy and committed relationship snapshot justify a reusable, agent-bound capability [24]. The following review explains that progression and the choices it produced.

3.2 Relevant Literature and Existing Systems

3.2.1 Encrypted Search and Whole-Workflow Disclosure

Searchable symmetric encryption was an early candidate for private data access. Li et al.'s survey organizes the field by query expressiveness, including keyword, Boolean, and range searches [17]. It distinguishes search patterns, access patterns, and leakage from updates, then reviews attacks and defenses. This taxonomy helped us separate encrypted storage from information revealed through its use. It gave our initial review a structure for query functionality and observable leakage.

Zhang et al. combine Word2Vec-based keyword expansion with inner-product encryption for semantic multi-keyword search [32]. This connects a query's meaning to encrypted retrieval. Our later question concerned the authority to perform that retrieval: which agent, task, action, and resource scope should an enforcement point accept? Search matching and authorization answer different parts of an agent's data-access task.

Gui et al. examine leakage across the encrypted index and document-retrieval workflow [14]. Their analysis directs attention beyond an isolated search operation. Du and Liu examine a different exposure: record counts, table sizes, and file growth can reveal business information without decrypting records [9]. They propose artificial records to obscure these aggregate observations. Together, these works motivated our attention to what an access workflow reveals through its outputs and repeated use.

This reading shaped the separation between Raziel and the broader Hypha vision. We focused Raziel on checkable authority and private relationship witnesses. Confidential retrieval, dataset relevance, and cumulative disclosure concern data selection and use. Their distinct evidence requirements explain the pivot from an encrypted-search system to proof-carrying authorization.

3.2.2 Cryptographic Access and Computation on Encrypted Data

Ciphertext-policy ABE places an access structure in a ciphertext. A user decrypts through keys associated with attributes that satisfy that structure [5]. This gives cryptography a direct role in controlling access to content. It raised a useful design question for us: should permission be expressed through decryption keys, or checked as a separate authorization decision?

Chiquito et al. survey ABAC and ABE for industrial data sharing, then propose combining their policy-management and decryption functions [7]. A trusted policy decision point supplies information for encryption and key generation. Their discussion connects changing policy to key custody, issuance, and revocation. This helped us separate policy administration, authorization evaluation, and enforcement rather than treating encryption as the entire access system.

Goldwasser et al. extend ABE, functional encryption, and reusable garbling to Turing-machine computation [13]. Their function keys depend on program-description size rather than its running time. This reading broadened our interest from encrypted matching to expressive computations over private inputs. Our implementation takes a zkVM route: one fixed guest interprets policy inputs, so a new policy changes the input rather than requiring a new proof program.

3.2.3 Private Attribute-Based Authorization

Kerl et al. provided the closest starting point for our private-authorization question [16]. Their protocol uses attributes certified by trusted third parties and encrypted under the requester's Paillier public key. The data owner holds the signed ciphertexts and computes comparisons against its policies. The requester decrypts a masked challenge response, which the owner checks to determine access. The intended privacy boundary protects attribute values from the data owner and plaintext policies from the requester.

This is an interactive authorization exchange. Its basic comparison tests equality. Conjunction combines attribute tests, and separate policies express disjunction. The design

separates policy changes from the certified encrypted attributes already held by the owner. This made private evaluation over changing rules a practical starting point for our own work.

We changed the authorization statement and the evidence delivered after evaluation. Raziel evaluates supported relationship rules over a committed snapshot, including team membership, alternative paths, intersections, and typed caveats. The verifier receives approved PolicyIR, and the relationship witness stays private. A non-interactive proof supports a reusable, key-bound capability. Subsequent protected requests use fresh signatures within that grant. These choices connect relationship-aware evaluation to private evidence and reusable enforcement.

Kerl et al. discuss inference from repeated policy and response observations. This reinforced our decision to treat private evaluation and cumulative disclosure as separate questions. Raziel supplies evidence of snapshot authorization. Hypha's wider research concerns how repeated permitted access affects disclosure.

3.2.4 Relationship Policies and Graph Enforcement

Zanzibar's relationship tuples and userset operations informed our policy model. Its consistency controls connect permissions to changing content [20]. We used SpiceDB's caveats and revision-aware reads [2]. Its `ALLOW` or no-permission responses keep the witness private by default. A gateway can trust that response without receiving the graph. Tracing is opt-in [3, 4].

Mohamed et al.'s XACML4G expresses authorization through constraints on graph vertices, edges, and paths [18]. Rules can specify edge direction and minimum or maximum path length. Its architecture separates policy decisions from enforcement and evaluates path conditions through generated graph queries. A source-subset graph supplies authorization-relevant data for supported source mappings. This gives a useful example of graph-aware policy outside the Zanzibar model.

Their later comparison examines access-control methods for property-graph data [19]. It distinguishes systems returning permission decisions from systems returning filtered query results. It compares policy models, negative permissions, and datastore dependence. These axes help separate policy expressiveness from the place and form of enforcement. The papers protect graph-structured data. Raziel uses relationship facts to authorize operations on datasets. We retain that distinction in the comparison.

We used SpiceDB's relationship model and added portable proof of approved-policy evaluation over a committed snapshot. Our verifier checks the computation independently of the authorization engine and receives no private relationship witness [24].

3.2.5 Proof-Carrying Evidence and Reusable Authority

We adopted independent checking from proof-carrying authentication [1] and its application to authorization in Garg's teaching note [12]. Logical derivations can expose their premises.

We replaced a disclosed derivation with a zkVM proof over an owner-held relationship witness. This retains checkable evidence and keeps the relationship witness private.

We combined reusable authority with request-level key binding. Macaroons restrict delegated bearer credentials through caveats [6]. DPoP binds OAuth token use to a presenter key and signed request proofs [10]. These established patterns informed our scope, expiry, and signature controls. Raziel uses its own request format and adds proof of committed relationship evaluation.

3.2.6 Verifiable Execution and Privacy

We implemented authorization evaluation on SP1 using the recorded SP1 6.3.1 PLONK profile [24]. Pinocchio demonstrates publicly verifiable computation [21]. PLONK uses a universal, updatable structured reference string with setup trust [11]. SP1 distinguishes zero-knowledge PLONK and Groth16 wrappers from individual STARK proofs [30]. We used execute checks for functional evaluation and PLONK proofs for cryptographic evidence.

ZK-RAM links memory representation to proving cost [31]. Its VOLE-based RAM results motivate further study of interpreter cost. Our SP1 measurements retain their own workload and machine profiles.

Table 3.1: Research basis for our evidence objects and trust boundaries

Research family	Evidence or protected object	Design lesson for Raziel
Encrypted search	Query matching, access patterns, and retrieval	Whole-workflow disclosure analysis.
ABE	Attribute-governed decryption	Distinct content and policy controls.
Private ABAC	Evaluation over encrypted attribute values	Explicit input-visibility boundaries.
Logical PCA	Checkable policy derivation	Independent checking.
Zanzibar / SpiceDB	Revision-aware relationship decision	Revision-bound relationship state.
Graph-policy systems	Paths, decisions, and filtered graph results	Distinct policy and enforcement choices.
Macaroons / PoP	Restricted credential and presenter key	Scoped, key-bound use.
Verifiable execution	Program execution and public outputs	Proof of approved-policy evaluation.
Raziel	Positive authorization for capability scope	Combines approved policy, committed state, verification, and reuse.

3.3 Project Development and Evaluation

We built and evaluated the backend, public SDK, operator console, and agent applications through repeated implementation and testing. This follows Hevner et al.'s artifact-and-evaluation method [15]. Labelled synthetic ledgers supported the complete reconciliation workflow, from access request to returned settlement evidence [24].

We evaluated the questions in Chapter 1 through three sets of checks. For RQ1, we compared the supported semantics, checked deterministic vectors, and generated real local proofs. For RQ2, we inspected public fields, rejected mutations, and verified a retained proof with the issuer and prover stopped. For RQ3, we tested signed SDK requests and cold and cached protected queries. Shared proof-critical evaluation code kept host and guest semantics aligned. Separate tests covered semantic agreement on the implemented fragment, proof validity, and gateway enforcement.

3.4 Evaluation Measures

Our evaluation recorded guest instructions, execution and proving time, proof size, verification time, peak memory, and completion or failure. Our records retain program and key identifiers, proof mode, versions, CPU allocation, memory limits, and dataset provenance. We reported raw proof bytes separately from package size. These run-specific records explain local proving feasibility and the resources used.

We treated queue delay, issuance, cold verification, cached verification reuse, and protected-query execution as separate cost categories. This connects the measured issuance and verification costs to our reusable-capability design. Each cached request retained authentication, scope, expiry, and replay checks.

Our disclosure review covered public values, responses, receipts, and approved logs. The review treated outcomes, timing, public scope, linkability, and permitted outputs as observable information [14, 30]. This kept witness protection distinct from inference through authorized access.

Chapter 4

System Design and Protocol

4.1 Authorization Statement and System Boundary

We implemented proof-carrying authorization with separate decision, proof, verification, and data-enforcement stages. The data owner publishes an approved policy and commits a relationship snapshot. The prover constructs a private witness for one permitted request. An independent verifier checks the proof and public bindings. The gateway returns data only after request, scope, and dataset checks pass.

Policy and snapshot authority signatures establish trusted provenance. The computation proof establishes evaluation under those inputs [24].

Let P denote a canonical policy, w a private witness with Merkle membership openings, h_P the expected policy hash, r_G the relationship snapshot root, and q the capability scope. For the fixed bindings h_P, r_G, and q, we prove that there exist a policy P and witness w satisfying all three checks below.

Policy identity.

$$H_P(P) = h_P. \tag{4.1a}$$

The canonical hash of the supplied policy must match the expected policy hash. This binds the computation to that policy.

Snapshot membership.

$$\mathrm{Open}(w, r_G) = 1. \tag{4.1b}$$

The Merkle openings must show that the relationship evidence belongs to the committed snapshot. Here, 1 means that the membership check passes. The verifier checks the proof without receiving those private relationships or their openings.

Authorization evaluation.

$$\mathrm{Eval}(P, w, q) = \mathrm{ALLOW}. \tag{4.1c}$$

The policy interpreter must return `ALLOW` for the exact capability scope q, using the validated witness w. Snapshot membership alone does not grant access. The evidence must satisfy the

policy for that scope.

The guest binds request commitment, scope hash, issuance time, expiry, and nullifier to this relation. Service boundaries check policy signatures, registry status, snapshot attestations, and request signatures.

4.2 Architecture and Trust Zones

We separated requesting agents, data-owner services, and independent enforcement (Figure 4.1). The operator controls approval and relationship administration. Agents control their request signing keys. The prover reads private authorization inputs. The verifier receives the approved policy and public evidence through a local trusted registry.

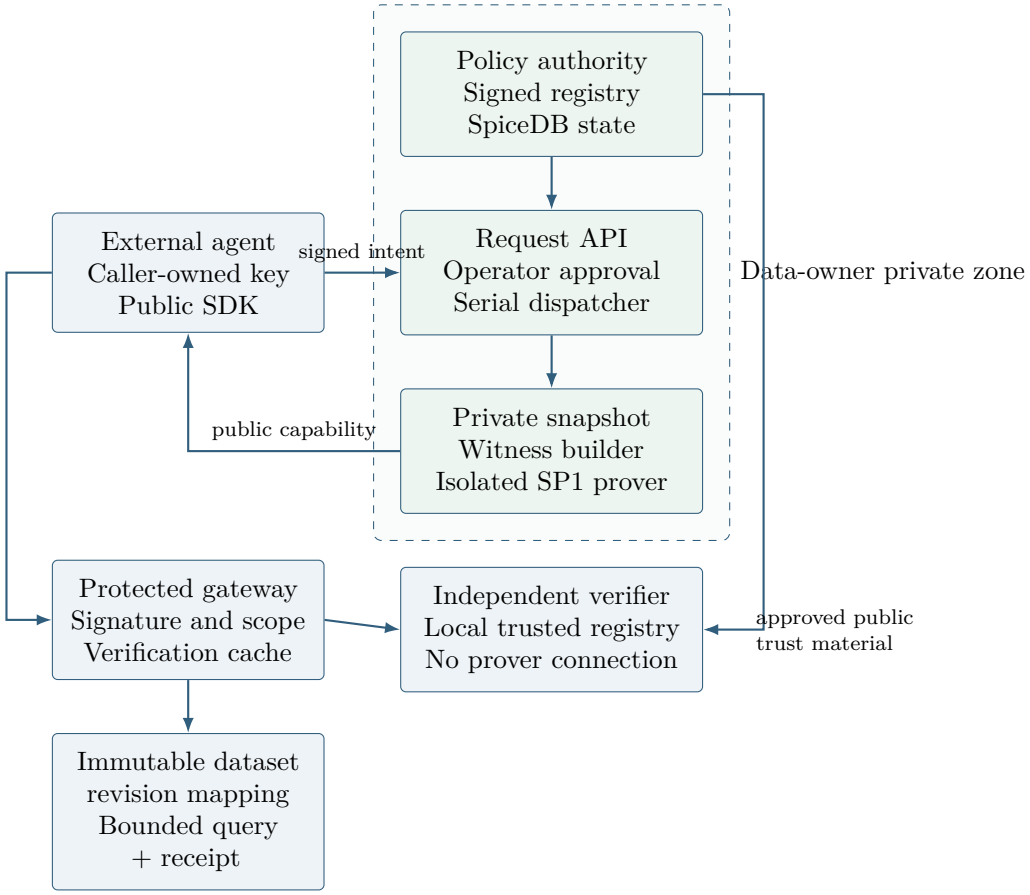


Figure 4.1: Component and trust-boundary view. The verifier has no connection to the prover service. The public capability crosses the owner boundary, but the private witness does not.

We enforced these boundaries through separate processes, networks, filesystem mounts, and keys. The policy authority signs approved records. The snapshot authority signs relationship descriptors. The registry writer publishes validated generations. Policy approval remains outside the API and proof worker.

We kept tuples, stored caveat values, satisfying paths, salts, and Merkle openings owner-side. Agents receive public scope, policy hash, snapshot root, validity, and proof or receipt.

The verifier receives the approved PolicyIR, and agents observe access outcomes. Our privacy boundary protects the relationship witness from both.

4.3 From a Visual Policy to Approved PolicyIR

We connected the visual policy editor to a Go adapter for maintained SpiceDB compiler output. Rust converts that output into PolicyIR and rejects unsupported constructs.

The interpreter supports direct relations, subject sets, union, intersection, one bounded arrow, and typed caveats over strings, integers, and Booleans. Five UI families express this subset: direct, team, alternative paths, dual control, and caveated access. All five families use the same fixed guest, so policy changes update its input.

A canonical policy is committed as

$$h_P = H\big(\texttt{raziel/policy/v1}, \mathrm{Canon}(P)\big). \tag{4.2}$$

Canonical encoding fixes ordering, types, versioning, and framing. We hash canonical policy content rather than raw Protobuf bytes. A family groups rules. Installed version identifies a namespaced package. Approval epoch orders approvals for a policy name. Registry generation identifies published registry state. The hash commits policy content, keeping reapproval distinct from content changes.

4.4 Relationship State and Snapshot-Time Authorization

We separated reusable policy definitions from relationship facts, such as a dataset's team and an agent's membership. Operators manage these facts independently of policy publication. Managed-mode issuance evaluates existing relationships and preserves missing-permission failures.

We captured one SpiceDB revision in a canonically ordered, blinded Merkle tree. A private salt limits guessing over small relationship sets. Openings bind tuples and indices. The private leaf count fixes tree shape. The signed descriptor binds the root, approved policy, revision, epoch, validity, trust domain, encoding, and signing identity.

We implemented positive, `ALLOW`-only proofs. The host finds a satisfying witness. The guest checks openings, evaluates the policy, and rejects unused or ambiguous witness material. Intersection requires evidence for every required branch. Union requires a satisfying supported branch. A missing required relationship yields no capability. Denial remains a service outcome rather than a zero-knowledge proof.

We bound each grant to its policy hash and relationship root. A separate trusted dataset mapping fixes the permitted data revision. Together these bindings preserve the authorization context and snapshot coherence.

Snapshot coherence preserves the issued version, rather than certifying current freshness. Removing membership leaves an earlier snapshot and its proof unchanged. The baseline

cache can retain an admitted capability until its short expiry. Immediate revocation requires an explicit freshness check or invalidation mechanism.

4.5 Asynchronous Issuance and Agent Consent

We separated proof generation from the routine request path. An access intent arrives through the SDK. The operator selects an approved policy and exact dataset binding. The dispatcher reserves capacity, prepares the private snapshot, and offers a snapshot-bound issuance proposal. The agent signs that exact proposal before the expensive proof job is submitted.

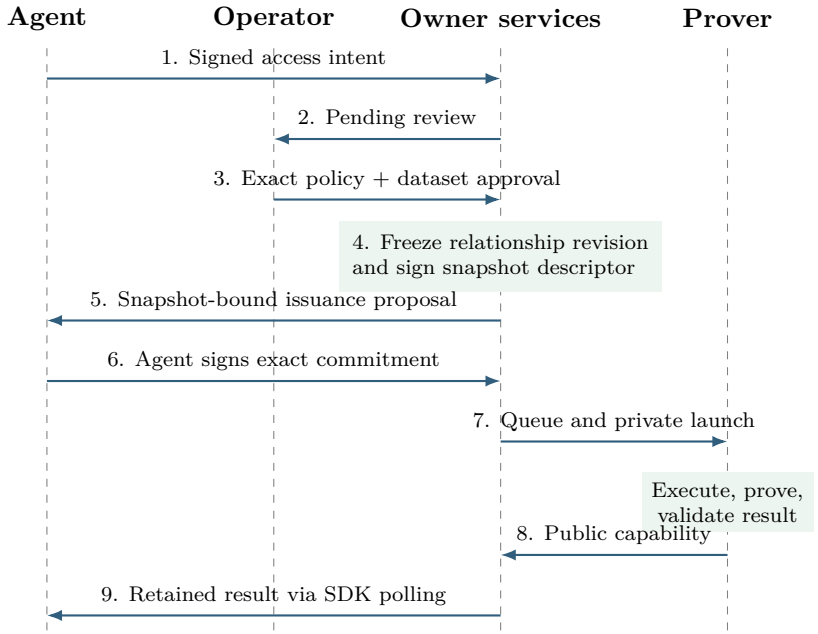


Figure 4.2: Access-intent approval and proof issuance. Human review and the final agent signature precede the isolated proof job. Polling retrieves the public result.

The final proposal binds the agent's consent to the exact snapshot and validity. The SDK signs through a caller-owned interface without transferring key custody.

The commitment binds agent key, task, purpose, action, resource type and scope, expiry, nonce, snapshot identity, and typed context. Public proof values bind policy hash, relationship root, request commitment, `ALLOW`, scope hash, issue/expiry times, and nullifier. These prevent relabeling a proof for another task.

We used BullMQ over owner-only Redis to connect the Rust API and proof supervisor. Queue messages carry bounded orchestration state. Private envelopes and immutable snapshots remain in separate owner-only storage. We ran one job at a time under the measured VM resource profile. The isolated prover has no network path to a hosted service. This keeps private authorization inputs within the data-owner deployment.

4.6 Protected Requests and Safe Reuse

We made capabilities reusable and protected each API request against replay. Every protected request carries a fresh nonce, timestamp, target, body hash, method, public key, and signature. The gateway derives the key thumbprint and matches it to the capability. It checks action and resource scope before reading data.

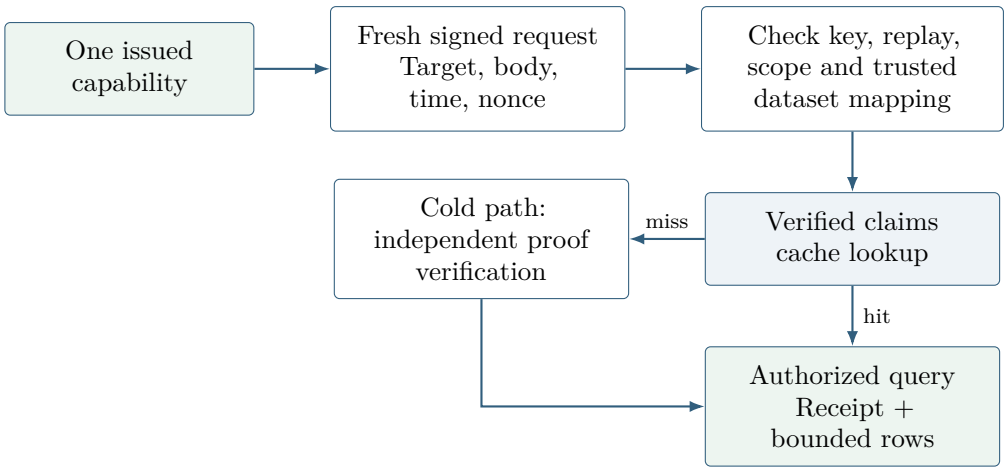


Figure 4.3: Cold verification and capability reuse. A cache hit skips repeated proof verification, not request authentication, replay defense, or scope enforcement.

On a cold request, our independent verifier checks the proof and public claims against its trusted registry. It runs without contacting the issuer or prover. A verified projection then enters the cache. Later requests reuse it with fresh signatures, key binding, scope, expiry, and replay checks. Only admitted requests reach the data adapter.

We resolved both external-agent paths through an immutable trusted mapping from policy/type/scope to dataset identity, revision, and checksum. The relationship root commits authorization tuples. The mapping binds the data revision and checksum. A revision-1 capability remains confined to revision 1. This blocks the cross-version retargeting pattern associated with the New Enemy problem on these routes. Zanzibar-style consistency tokens set a minimum authorization revision for a content operation [2].

Our proof establishes scoped permission under trusted policy and snapshot authorities. Gateway enforcement and key protection complete the access boundary.

Chapter 5

Implementation and User Interface

5.1 Repository and Language Boundaries

We split the implementation across three repositories so external agents can use a public SDK without importing privileged backend code. The repositories contain backend enforcement, the SDK, and the two agent applications. Protobuf and generated clients carry wire contracts. Rust owns canonical commitment rules.

Table 5.1: Implemented component ownership

Component	Technology	Responsibility
Console and gateway	TypeScript, Next.js	Administration, review, protected routes, receipts, verification cache.
Canonical and proof core	Rust	PolicyIR, encodings, Merkle checks, witness validation, SP1 host/guest.
Compiler adapter	Go	Pinned SpiceDB compiler output to neutral input.
Request and proof services	Rust, BullMQ, Redis	Durable requests, bounded admission, serial dispatch, supervision.
Relationship store	SpiceDB, PostgreSQL	Permission schema and durable facts.
Public SDK	TypeScript	Signing interface, issuance, polling, queries.
Agent A and Agent B	TypeScript	Separate identities, task state, business tools, bounded messages.

The compiler rejects unsupported constructs. A fixed SP1 interpreter proves `ALLOW` only. Guest changes require compatible trusted verification keys. The independent verifier receives approved PolicyIR and public evidence, not private relationships or openings. Agents receive neither PolicyIR nor the witness.

5.2 Operator Interface

We built dataset controls for editing, validation, finalization, and cloning. Finalized revisions retain row counts, checksums, and schema fingerprints. Our policy builder compiles supported

blocks for review and activation. The catalogue distinguishes Version ID, Family ID, approval epoch, and hash. Role administration shows full installed Version IDs. Cloning preserves the finalized revision and opens an editable copy.

Our Agents workspace registers public keys and issues API tokens. Managed policies expose direct-role and team assignments. Request review binds agent, task, purpose, action, dataset revision, policy, and proof-job identity. Receipts expose public audit records and preserve witness privacy. API tokens grant service entry. Signed requests require proof of possession from the registered signer.

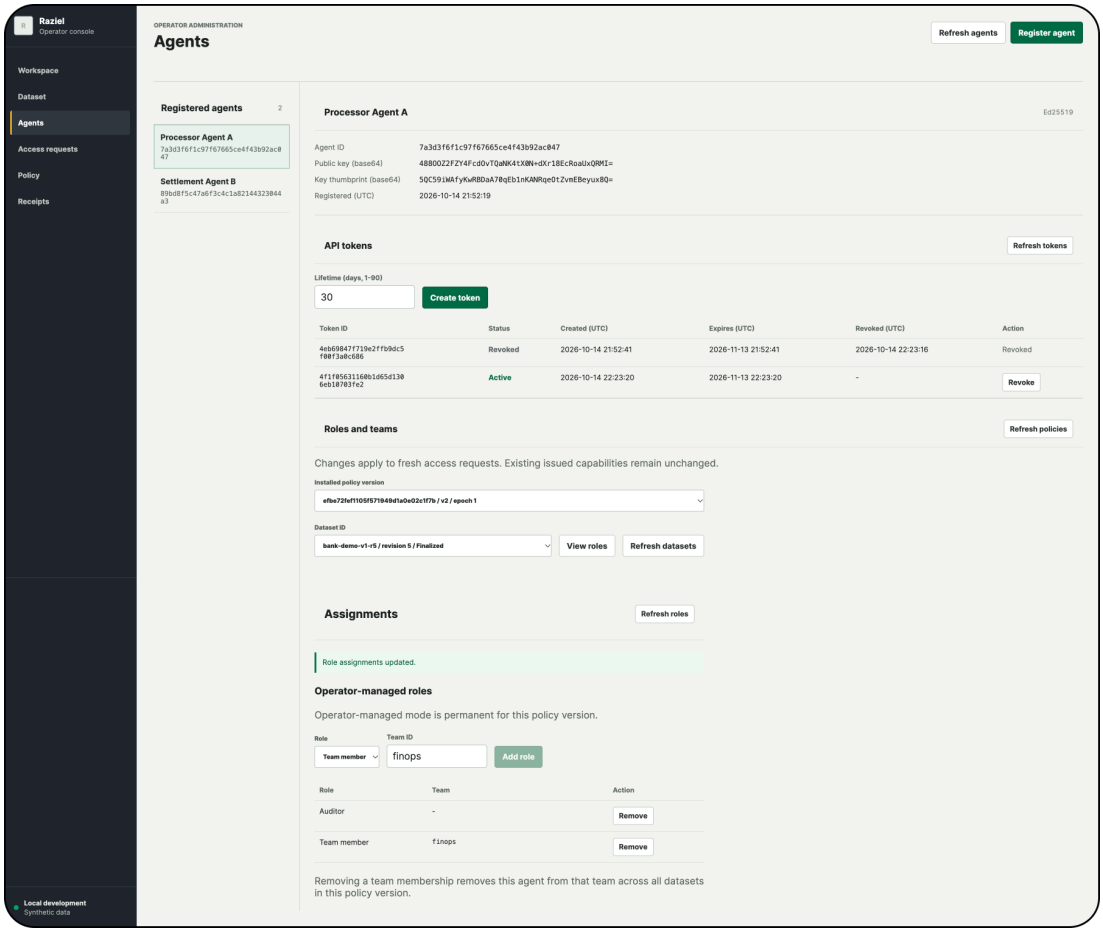


Figure 5.1: Operator policy and role administration. The installed Version ID, immutable synthetic dataset revision, supported role, and registered agent assignment identify the operator's authorization choice.

5.3 Public SDK and External Agent Consumption

We published `@gitaalekhyapaul/raziel-sdk-ts@0.1.0` for caller-owned public keys and asynchronous signers. Our client creates, reads, and resumes its own requests, signs exact issuance proposals, polls within bounds, and submits protected queries. It rejects unknown versions, fields, and enum values. Remote proof jobs survive local timeouts. Our dashboards retain task and signing-attempt state. Exact retries reuse request identity and the existing proof

job.

The gateway maps authorized resource scope to an immutable dataset revision and checksum. The relationship root commits authorization facts, not ledger contents.

5.4 Agent Tools and the LLM Boundary

Agent A handles processor transactions. Agent B handles bank settlements. Each retains its own key and capability. Their bounded channel exchanges requests and results, not authority.

Our optional OpenAI-compatible adapter limits execution to four loop turns, four calls per response, and a 30-second completion request timeout. Validated tools calculate amounts, and manual controls retain a deterministic task path. The adapter validates tool definitions, bounds response size, and treats ledger fields and messages as data. An LLM selects tools and summarizes results. Policy approval stays with the operator. SDK and gateway checks govern every protected request.

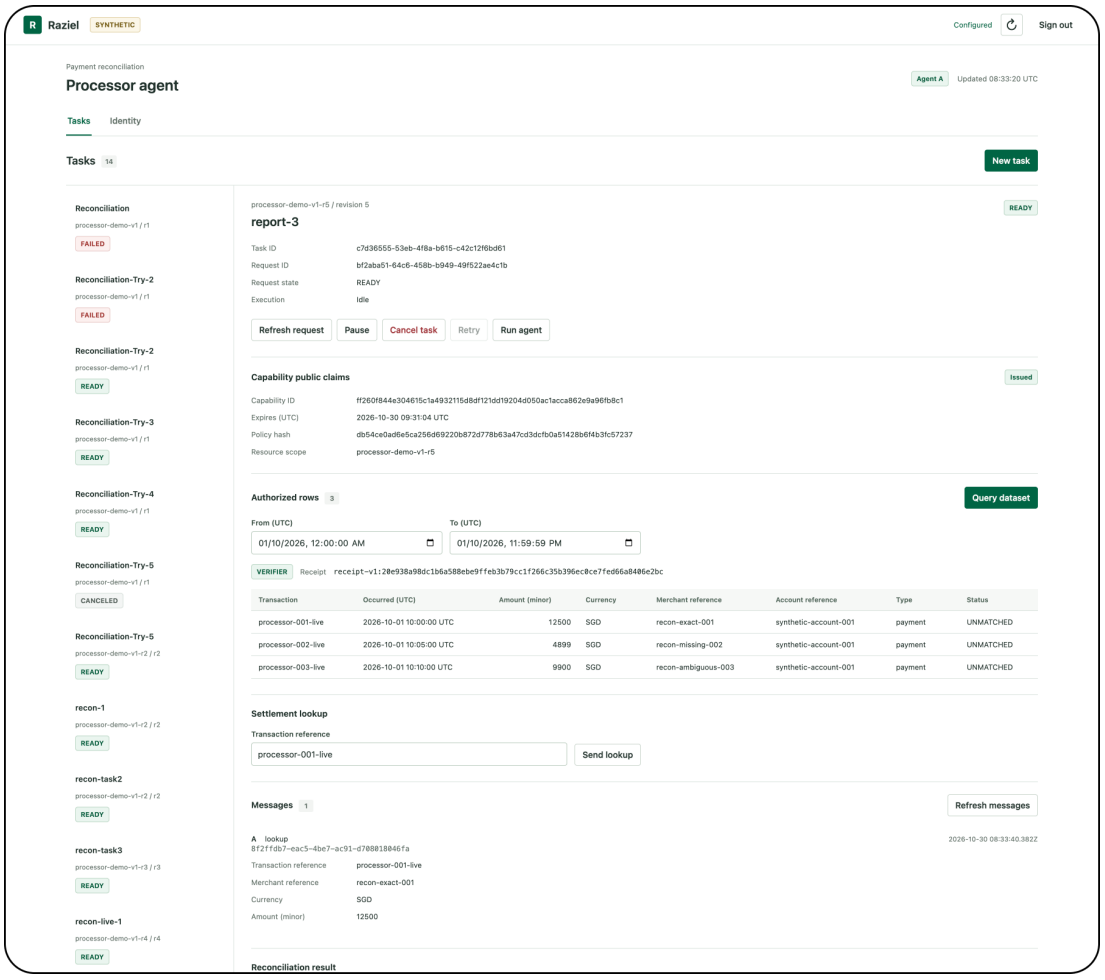


Figure 5.2: Agent A capability and processor query. The public capability scope binds the agent's processor revision. Authorized synthetic rows and a receipt show the protected-query result.

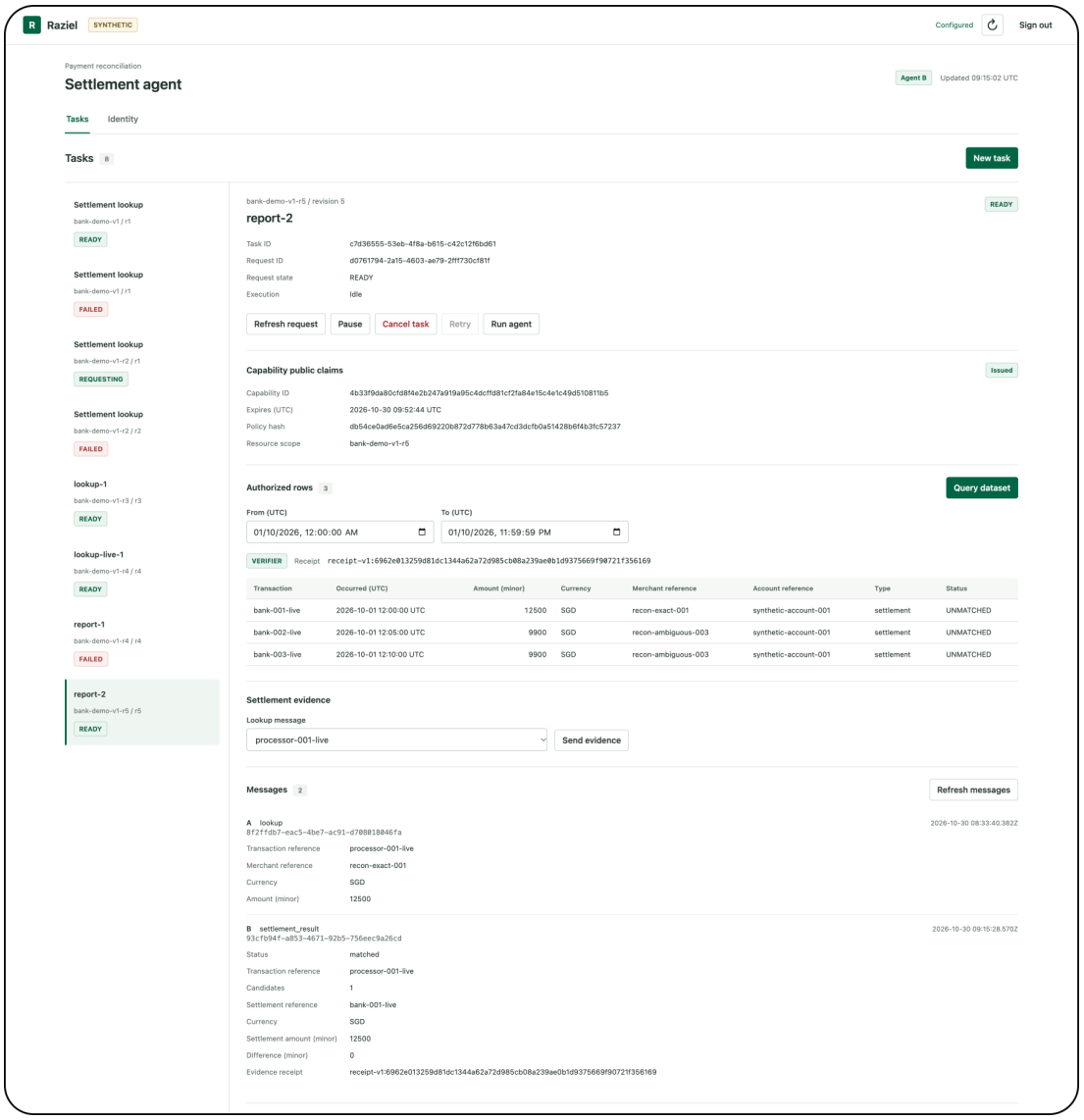


Figure 5.3: Agent B settlement evidence. A separate bank capability supports the settlement query. The shared task/reference links the bounded synthetic result to Agent A.

5.5 Container Deployment and Operational Controls

Docker Compose separates owner services, gateway, dispatcher, and independent verifier. Owner-only storage holds snapshots and private envelopes. The proof sandbox has a read-only root filesystem and no network. The proving VM supplies real local proofs. SSH forwarding serves dashboards. Docker-managed ngrok supplies fixed HTTPS ingress for external SDK requests. Third-party HTTPS termination remains a demo trust assumption. These controls supplement cryptographic checks. Separate verifier dependencies avoid loading the full prover SDK and native proving runtime. Public proving setup and approved guest material use read-only permissions.

Chapter 6

Two-Agent Payment Reconciliation

6.1 Business Task and Synthetic Ledgers

Payment reconciliation requires evidence that a processor transaction corresponds to a bank settlement. We use this task to examine collaboration between agents with separate data-access scopes. Agent A reviews processor transactions. Agent B retrieves bank settlement evidence. Each agent accesses its ledger through its own capability. The operator administers the labeled synthetic datasets and their access policies.

We reconcile reference, currency, and amount using integer minor units. Our tools distinguish exact matches, missing settlements, and ambiguous candidates. A match requires agreement across all three fields. Editing or cloning synthetic records changes the task inputs. Reference, currency, amount, and settlement status present the business outcome independently of proof-system details.

Table 6.1: Illustrative synthetic reconciliation cases

Ledger	Shared reference	Amount	Interpretation
Processor	DEMO-1042	SGD 125.00	Requires settlement evidence.
Bank	DEMO-1042	SGD 125.00	Exact reference/currency/amount candidate.
Processor	DEMO-1043	SGD 80.00	Missing settlement without a bank candidate.
Bank	DEMO-1044	Two candidates	Ambiguous result.

6.2 Operator-Supervised Agent Workflow

The operator finalizes the two dataset revisions and registers separate persistent agent keys and API tokens. Reusable policies define the permitted access paths. For operator-managed policies, role and team assignments remain separate from request approval. The operator can publish a policy or assign a role before an access request arrives, or during its pending review.

Agent A submits a processor-data access request through the SDK. The operator selects an installed policy version and the exact dataset revision. Approval starts snapshot preparation

using the existing relationships. Agent A signs the snapshot-bound proposal, and Raziel queues the authorization proof. Successful issuance returns a capability bound to A's key, task, and processor scope.

Agent A queries the permitted processor records and selects a transaction for settlement lookup. Its message to Agent B carries the transaction reference and shared task context, not A's capability. Agent B submits a separate bank-data access request. Its own approval and proof produce a bank-scoped capability. Agent B then queries the bank ledger and returns a bounded match result with an evidence reference.

Agent A combines the returned evidence with its processor record and displays the reconciliation status and amount difference. Each agent dashboard presents its own capability claims, authorized rows, messages, and receipts. The operator can inspect both access requests and their public audit records. The shared reference connects the results without merging agent identities or transferring authority to query the other ledger.

6.3 Authorization Controls in the Scenario

Managed direct-role and team assignments let the operator change authorization facts independently of the reconciliation tools. A fresh snapshot must contain the relationships required by the selected policy. Missing membership prevents capability issuance during authorization preparation, before expensive proof generation. Adding the required role allows a new request to use the updated relationship state.

Each issued capability retains its original policy hash and relationship root. Removing a role affects fresh snapshots. Existing capabilities retain their snapshot-time authority, subject to expiry and configured invalidation rules.

We exercised out-of-scope, replayed, expired-request, altered, and unsigned requests through the portal. These controls returned no transaction rows [25]. Every protected query requires a fresh request signature and nonce. The gateway checks the key binding, capability scope, and dataset revision before returning records.

6.4 Reconciliation Outcome and Capability Reuse

Agent A queried the authorized processor dataset and retrieved three records. Agent B used a separate capability to query the bank dataset and returned matching settlement evidence. This evidence identified synthetic bank record `bank-001-live`, an SGD 125.00 settlement, and an amount difference of zero. Both protected queries produced public receipts.

The result combines data obtained through two independent grants. The proof establishes authorization under committed relationship facts. The reconciliation tool compares the permitted records and calculates the business result. Public receipts connect each query to its agent, task, policy, and dataset revision.

After issuance, agents can repeat signed queries within the same capability scope without

Table 6.2: Observed two-agent reconciliation outcome

Stage	Result	Authorization context
Processor query	Three authorized records	Agent A's capability, selected finalized processor revision, public query receipt.
Bank query	Settlement evidence	Agent B's separate capability, selected finalized bank revision, public query receipt.
Returned result	`bank-001-live`, SGD 125.00, zero amount difference	Shared transaction reference and bank-query evidence receipt.



Figure 6.1: Reconciliation result and audit trail. Agent A displays the returned settlement match, amount difference, and bank-query evidence reference. Separate capabilities govern the two ledger queries.

generating another proof. A cached verification result removes repeated proof checking, but retains request authentication and scope checks. This separates the cost of establishing authority from the agent's subsequent work on the authorized ledger. Figure 6.2 brings together the actors and data flows. Figure 4.2 details the issuance protocol.

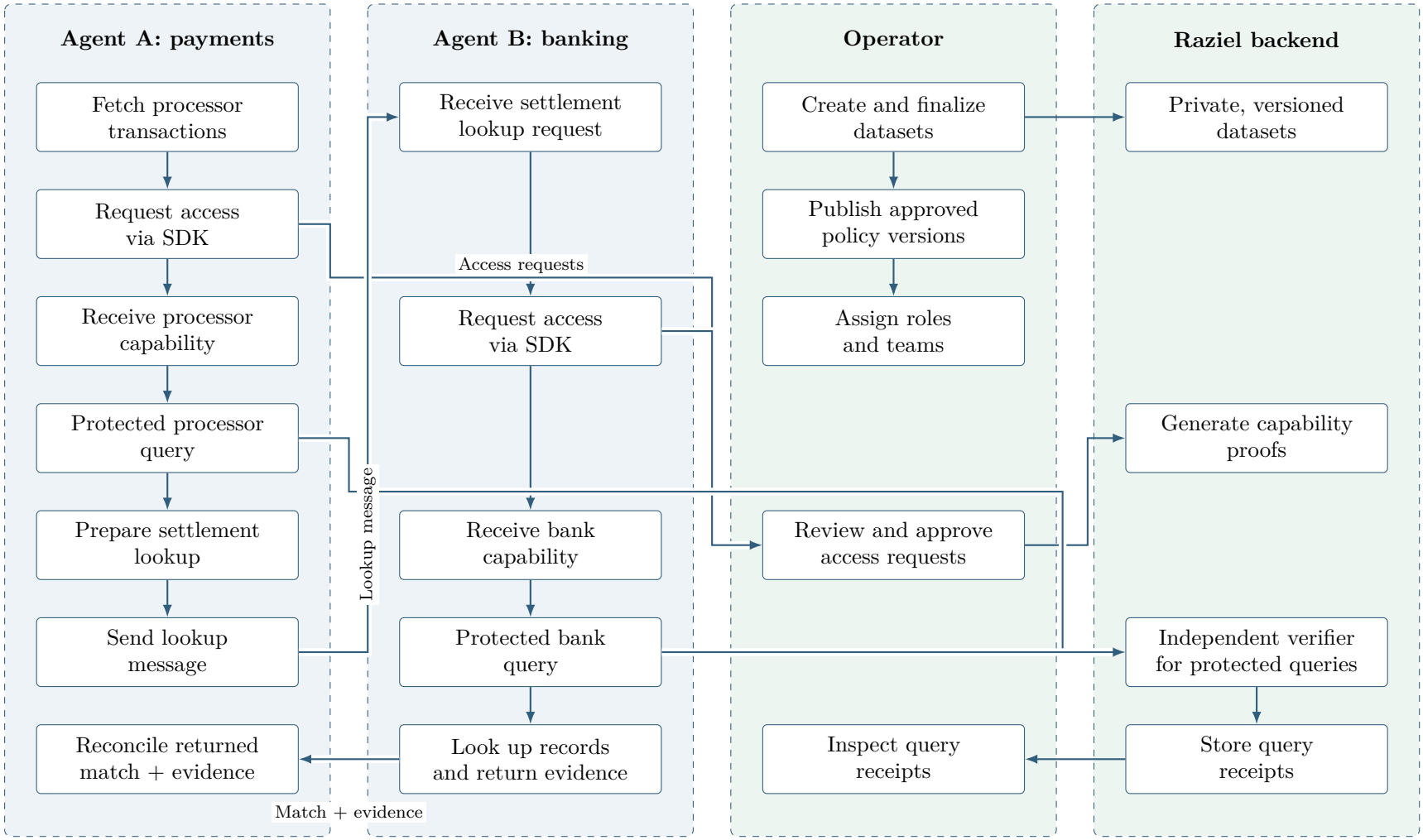


Figure 6.2: Two-agent reconciliation across separate capability scopes. Each agent receives its own capability after proof generation. Figure 4.2 specifies the approval, snapshot, signature, and proof ordering. Messages carry bounded business evidence, not data-access authority.

Chapter 7

Evaluation and Findings

7.1 Evaluation Design

We evaluated canonicalization, supported semantics, real local proofs, and protected access through protocol tests, VM experiments, and the reconciliation case. Synthetic ledgers record revisions, checksums, schema fingerprints, and row counts. Performance values describe individual runs. We measured proof and verification durations from benchmark records. The reconciliation case evaluates integration and protected-access behavior.

The August 13 transcript records 16 host CPUs and 62.79 GiB of memory. The eight-CPU container profile has a 56 GiB hard limit and no swap. Go and Rayon concurrency are eight, and the Go memory target is 24 GiB. Service and portal CPU counts follow deployment profiles. The pinned environment uses SP1 6.3.1 and Rust 1.97.1. Appendix A identifies each experiment's source record, including differing commits and keys.

7.2 Proof Generation and Resource Results

We report each proof run separately in Table 7.1. Times use seconds, GiB uses 2^{30} bytes, and artifact sizes use bytes.

Table 7.1: Individual successful real-Plonk runs

Run	CPUs	Prove (s)	Peak (GiB)	Bytes	Record
Direct, 10 Aug	4	1742.106	43.12	2537	Public cluster record
Direct, 13 Aug	8	929.660	42.46	2537	Recorded cluster transcript
Retained, 25 Aug	8	932.309	43.09	2536	Paired verifier fixture
Service, 26 Aug	8	955.516	43.77	2536	Full API/job/proof path
Portal, 30 Aug	8	937.078	—	2536	Live operator walkthrough

The direct runs took about 29.04 minutes on four CPUs and 15.49 minutes on eight. Their ratio is about 1.87, a 46.6% elapsed-time reduction. Source-manifest digest and verification-key identity match. The eight-CPU transcript omits its source commit and toolchain image identity, so this is not a controlled scaling study [22, 23].

Our service run produced a 2,536-byte proof in 955.516 seconds, against 458 milliseconds of guest execution. Peak memory reached 43.77 GiB with no swap use or OOM event [26]. The portal experiment connected request submission, operator approval, asynchronous issuance, and a protected query [25].

7.3 Independent Verification and Mutation Results

We verified the retained proof in 447 milliseconds with every issuer/prover role stopped. The worker had no network, a read-only root, a 2 GiB limit, no swap, and a 7.77 MiB observed peak [27]. The direct eight-CPU record reports 438 milliseconds for a different artifact and key. Both runs demonstrate subsecond proof verification.

Table 7.2: Negative checks recorded by the isolated verifier and live gateway

Mutation/control	Observed result	Boundary established
Proof or public journal changed	Reject	Cryptographic evidence is required.
Verification-key identity changed	Reject	Trusted program identity is checked.
Policy or snapshot root changed	Reject	Inputs cannot be relabeled.
Scope or expiry changed	Reject	Capability claims are checked.
Request signature changed	Reject	Exact request binding is required.
Wrong-key binding in service flow	Reject	Capabilities remain key-bound.
Out-of-scope, replayed, expired-request, altered, unsigned portal controls	No transaction rows	A valid proof does not bypass request checks.

We verified the service capability with issuer and prover stopped, then rejected an altered capability and a request signed with a different key.

7.4 Capability Reuse and Cost Interpretation

We demonstrated a cold verified query and a cache hit using the same capability with fresh signed requests. One proof supported both protected reads.

Let P denote proof generation, V cold verification, S_i request authentication and scope enforcement, and D_i the permitted data operation. For one capability and N valid queries:

$$T(N) = P + V + \sum_{i=1}^{N} (S_i + D_i). \tag{7.1}$$

The model assumes an acceptable cached projection, no cache eviction requiring another proof check, and no reissuance for changed scope or expiry. Amortized setup cost is $(P + V)/N$.

For the direct eight-CPU example, $P + V \approx 930.098$ seconds. Setup contributes 9.30

seconds per query at $N = 100$ and 0.93 seconds at $N = 1000$. These calculated setup costs favor repeated queries within task-scoped grants. They are separate from measured request latency.

7.5 Answers to the Research Questions

RQ1: Computational feasibility. Real Plonk proofs, independent verification, and mutation rejection establish feasibility for the supported `ALLOW` relation, committed state, and public scope.

RQ2: Privacy and trust separation. Private relationship witnesses stay owner-side. Agents receive scope and proof without graph access. The verifier receives the approved policy and public evidence.

RQ3: Practical integration and reuse. We connected the asynchronous API, published SDK, operator portal, agents, gateway, and receipts to protected access and reuse. The reconciliation case returned authorized records from both ledgers and an SGD 125.00 settlement match with zero difference and a linked receipt (Section 6.4). Capabilities bound each agent's exploration through signature, scope, and dataset-binding checks.

7.6 Operational Implications

Our VM experiments and reconciliation case support asynchronous authorization followed by repeated protected queries. Operators approve authority and queue local proofs. Agents reuse capabilities through signed, scope-checked requests. Authorization follows the committed snapshot, and the gateway binds dataset revision and checksum. Expiry and configured invalidation rules govern continued use after relationship changes.

Chapter 8

Conclusions and Recommendations

8.1 Project Outcomes

We built and evaluated an authorization pipeline connecting operator-approved policies to task-bound agent data access. Our implementation combines bounded policy evaluation, canonical commitments, private witnesses, real local Plonk proofs, and independent verification. Each agent receives a key-bound capability with checkable authorization evidence. We retain private relationship witnesses within the owner boundary and provide the approved policy to the verifier.

Our direct eight-CPU proof took 15.49 minutes. A separate isolated experiment verified a retained proof in 447 milliseconds with issuer and prover stopped. We published the SDK and connected the API/job flow, operator portal, and two agent dashboards. The two-agent reconciliation completed both protected ledger queries and returned a settlement match.

We demonstrated reusable computational evidence of snapshot authorization across protected queries. Cached requests retain fresh signatures, nonces, validity, scope, and trusted dataset checks. Our design separates expensive issuance from repeated data operations within the granted scope.

8.2 Future Work

Our results motivate four research directions.

8.2.1 Freshness and Revocation as a Measured Tradeoff

A study of snapshot lifetime, capability duration, cache retention, and revocation propagation can quantify the balance between reuse and changed relationship state. The proposed experiments compare batch, interactive, and emergency-revocation policies. Relevant measures include reissuance cost, stale-authorization exposure, query volume, and the time from a membership change to blocked access.

8.2.2 Comparative Performance and Larger Authorization Graphs

A matched comparison can test online relationship evaluation, signature-only capabilities, and proof-backed capabilities under the same workload. The proposed study varies graph size, supported path depth, policy shape, caveats, and reuse count. Repeated runs can separate CPU, memory, setup, and cache effects, with authorization checks measured apart from data-query cost.

8.2.3 From Positive Proofs to Private Denial

We see verifiable negative authorization as the next proof-system research question. The current system proves `ALLOW`. A future `DENY` proof must establish that no satisfying path exists within a complete committed state, not merely that one witness failed. Authenticated absence and bounded coverage are research requirements. Policy privacy from the verifier and recursive rules need separate proof models.

8.2.4 Data Provenance, Relevance, and Cumulative Disclosure

We propose distinct data commitments for content provenance and task relevance, separate from the relationship root. A study of repeated queries, aggregation, and cross-agent messages can measure cumulative disclosure. These questions extend our evaluated authorization boundary with separate evidence for data use and business conclusions.

8.3 Raziel Within the Hypha Vision

We position Raziel as one part of the wider runtime governance problem described in SAFR [28]. This project focuses on approved policies and authorization before data access. Our broader vision for Hypha brings authorization together with discovery, relevance, disclosure controls, and cross-institution accountability. Raziel provides the evaluated authorization subsystem for that vision. The other parts form the wider research programme for Hypha.

References

[1] Andrew W. Appel and Edward W. Felten. Proof-carrying authentication. In *Proceedings of the 6th ACM Conference on Computer and Communications Security*, pages 52–62. ACM, November 1999. doi: 10.1145/319709.319718. URL https://www.cs.princeton.edu/~appel/papers/says.pdf. Author manuscript dated 9 August 1999. Conference held 2–4 November 1999.

[2] AuthZed. SpiceDB documentation: Schema language, caveats, and consistency. First-party implementation documentation, n.d.. URL https://authzed.com/docs/spicedb/concepts/schema. See https://authzed.com/docs/spicedb/concepts/caveats and https://authzed.com/docs/spicedb/concepts/consistency. Accessed 21 October 2026. Living documentation.

[3] AuthZed. SpiceDB: Querying data. First-party implementation documentation, n.d.. URL https://authzed.com/docs/spicedb/concepts/querying-data. Permission checks and graph-query APIs. Accessed 23 October 2026.

[4] AuthZed. Validation, testing, debugging SpiceDB schemas. First-party implementation documentation, n.d.. URL https://authzed.com/docs/spicedb/modeling/validation-testing-debugging. Check Tracing section. Accessed 23 October 2026.

[5] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *2007 IEEE Symposium on Security and Privacy*, pages 321–334. IEEE, May 2007. doi: 10.1109/SP.2007.11. URL https://www.cs.ucla.edu/~sahai/work/web/2007%20Publications/SSP2007.pdf.

[6] Arnar Birgisson, Joe Gibbs Politz, Úlfar Erlingsson, Ankur Taly, Michael Vrable, and Mark Lentczner. Macaroons: Cookies with contextual caveats for decentralized authorization in the cloud. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. Internet Society, February 2014. doi: 10.14722/ndss.2014.23212. URL https://www.ndss-symposium.org/ndss2014/ndss-2014-programme/macaroons-cookies-contextual-caveats-decentralized-authorization-cloud/.

[7] Alex Chiquito, Ulf Bodin, and Olov Schelén. Attribute-based approaches for secure data sharing in industrial contexts. *IEEE Access*, 11:10180–10195, 2023. doi: 10.1109/ACCESS.2023.3240000. URL https://doi.org/10.1109/ACCESS.2023.3240000.

[8] Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. arXiv:2406.13352, version 3, 2024. URL https://arxiv.org/abs/2406.13352v3. Revised 24 November 2024.

[9] Genyuan Du and Jie Liu. Rethinking data security in aggregated databases: Beyond encryption. In *2024 9th International Conference on Signal and Image Processing (ICSIP)*, pages 526–530. IEEE, 2024. doi: 10.1109/ICSIP61881.2024.10671482. URL https://doi.org/10.1109/ICSIP61881.2024.10671482.

[10] Daniel Fett, Brian Campbell, John Bradley, Torsten Lodderstedt, Michael B. Jones, and David Waite. OAuth 2.0 demonstrating proof of possession (DPoP). RFC 9449, Internet Engineering Task Force, September 2023. URL https://www.rfc-editor.org/rfc/rfc9449.html. Standards Track. Sections 4, 7, and 11 distinguish token binding and request-proof checks.

[11] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Paper 2019/953, 2019. URL https://eprint.iacr.org/2019/953. First received 21 August 2019. Archive lists a latest revision dated 17 July 2025. Preprint.

[12] Deepak Garg. An introduction to proof-carrying authorization. Author-hosted teaching notes, December 2007. URL https://people.mpi-sws.org/~dg/papers/intro-pca.pdf. Version dated 11 December 2007. Original notes, not a peer-reviewed conference paper.

[13] Shafi Goldwasser, Yael Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run turing machines on encrypted data. Cryptology ePrint Archive, Paper 2013/229, 2013. URL https://eprint.iacr.org/2013/229. Author manuscript revised 9 June 2013. Published at CRYPTO 2013.

[14] Zichen Gui, Kenneth G. Paterson, and Sikhar Patranabis. Rethinking searchable symmetric encryption. In *2023 IEEE Symposium on Security and Privacy*, pages 1401–1418. IEEE, May 2023. doi: 10.1109/SP46215.2023.10179460. URL https://eprint.iacr.org/2021/879. Preprint revised 19 July 2022. Published in 2023.

[15] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS Quarterly*, 28(1):75–105, March 2004. doi: 10.2307/25148625. URL https://aisel.aisnet.org/misq/vol28/iss1/6/.

[16] Malte Kerl, Ulf Bodin, and Olov Schelén. Privacy-preserving attribute-based access control using homomorphic encryption. *Cybersecurity*, 8(1):5, 2025. doi: 10.1186/s42400-024-00323-8. URL https://link.springer.com/article/10.1186/s42400-024-00323-8. Article 5. Published 22 January 2025.

[17] Feng Li, Jianfeng Ma, Yinbin Miao, Ximeng Liu, Jianting Ning, and Robert H. Deng. A survey on searchable symmetric encryption. *ACM Computing Surveys*, 56(5):119:1–119:42, November 2023. doi: 10.1145/3617991. URL https://doi.org/10.1145/3617991. Article 119.

[18] Aya Mohamed, Dagmar Auer, Daniel Hofer, and Josef Küng. XACML extension for graphs: Flexible authorization policy specification and datastore-independent enforcement. arXiv:2306.12819, version 1, 2023. URL https://arxiv.org/abs/2306.12819v1. Extended version of an accepted SECRYPT 2023 paper.

[19] Aya Mohamed, Dagmar Auer, Daniel Hofer, and Josef Kueng. Comparison of access control approaches for graph-structured data. arXiv:2405.20762, version 1, 2024. URL https://arxiv.org/abs/2405.20762v1. Extended version of an accepted SECRYPT 2024 paper.

[20] Ruoming Pang, Ramon Caceres, Mike Burrows, Zhifeng Chen, Pratik Dave, Nathan Germer, Alexander Golynski, Kevin Graney, Nina Kang, Lea Kissner, Jeffrey L. Korn, Abhishek Parmar, Christina D. Richards, and Mengzhi Wang. Zanzibar: Google's consistent, global authorization system. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 33–46. USENIX Association, July 2019. ISBN 978-1-939133-03-8. URL https://www.usenix.org/conference/atc19/presentation/pang.

[21] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252.

IEEE, May 2013. doi: 10.1109/SP.2013.47. URL https://www.microsoft.com/en-us/research/publication/pinocchio-nearly-practical-verifiable-computation/.

[22] Raziel Project. SP1 Plonk Cluster Benchmark: cluster-20260810T174732Z-76324. Public experiment record, 10 August, 2026. URL https://github.com/gitaalekhyapaul/hypha-backend/blob/main/docs/benchmarks/sp1-plonk/cluster-20260810T174732Z-76324.md.

[23] Raziel Project. SP1 Plonk Cluster Benchmark: cluster-20260813T063649Z-8cpu. Maintainer-supplied terminal transcript summarized in repository, 2026. URL https://github.com/gitaalekhyapaul/hypha-backend/blob/main/docs/benchmarks/sp1-plonk/cluster-20260813T063649Z-8cpu.md. 13 August.

[24] Raziel Project. Raziel research software and experiment records. Three source repositories and dated benchmark records, 2026. URL https://github.com/gitaalekhyapaul/hypha-backend. SDK: https://github.com/gitaalekhyapaul/raziel-sdk-ts. Agents: https://github.com/gitaalekhyapaul/raziel-demo-agents. Benchmark: https://github.com/gitaalekhyapaul/hypha-backend/blob/main/docs/benchmarks/sp1-plonk/cluster-20260825T182353Z-18591.md.

[25] Raziel Project. Local Portal: Live VM Evidence and Screenshot Inventory. Repository functional evaluation record, 30 August, 2026.

[26] Raziel Project. Complete Service Flow: Authorization Experiment, 26 August. Public service-flow experiment record, 26 August, 2026.

[27] Raziel Project. SP1 Retained-Proof Verifier: verifier-20260825T192214Z-34749. Public isolated-verifier experiment record, 25 August, 2026. URL https://github.com/gitaalekhyapaul/hypha-backend/blob/main/docs/benchmarks/sp1-verifier/verifier-20260825T192214Z-34749.md.

[28] SAFR. Safeguards for agentic finance at runtime. Industry white paper, Version 1.1, October 2026. URL https://www.mas.gov.sg/-/media/mas-media-library/development/fintech/ai-safr/safr-v1-1.pdf. Version dated 9 October 2026. Accessed 23 October 2026.

[29] Herbert A. Simon. A behavioral model of rational choice. *The Quarterly Journal of Economics*, 69(1):99–118, February 1955. doi: 10.2307/1884852. URL https://doi.org/10.2307/1884852.

[30] Succinct Labs. SP1 documentation: Security model and proof types. First-party implementation documentation, n.d. URL https://docs.succinct.xyz/docs/sp1/security/security-model. See https://docs.succinct.xyz/docs/sp1/generating-proofs/proof-types. Accessed 21 October 2026. Living documentation. Project experiments pin SP1 6.3.1.

[31] Yibin Yang and David Heath. Two shuffles make a RAM: Improved constant overhead zero knowledge RAM. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 1435–1452. USENIX Association, August 2024. ISBN 978-1-939133-44-1. URL https://www.usenix.org/conference/usenixsecurity24/presentation/yang-yibin.

[32] Yu Zhang, Yifan Wang, and Yin Li. Searchable public key encryption supporting semantic multi-keywords search. *IEEE Access*, 7:122078–122090, 2019. doi: 10.1109/ACCESS.2019.2937846. URL https://doi.org/10.1109/ACCESS.2019.2937846. Published 27 August 2019.

Appendix A

Reproducibility and Evidence Index

A.1 Implementation Revisions

We reviewed the implementation repositories on 21 October 2026 and rechecked their revisions on 29 October 2026. Table A.1 identifies the current local `main` heads. All three working trees were clean.

Table A.1: Locally verified implementation revisions, 29 October 2026

Repository	Full commit identity
Backend	`1019c67acf8d756889a46fef1a48470349d3cfb4`
Demo agents	`cf96231e30184b85e62e92b95d8ae31390887555`
SDK	`6239fad9d02b1047806b3c449eb593ea7cd9f119`

The earlier service and retained-verifier experiments used `41035c2` and `35db270`. Each measurement retains its recorded source revision. Benchmark records supply timing and resource measurements. Functional records document request handling, protected queries, and enforcement outcomes.

A.2 Experiment Records

These paths are relative to the backend repository. Full public records retain the relevant hashes and trusted identifiers.

Table A.2: Experiment records and source artifacts

Experiment	Type	Source record
Four-CPU direct	Benchmark	`docs/benchmarks/sp1-plonk/cluster-20260810T174732Z-76324.md`
Eight-CPU direct	Benchmark	`docs/benchmarks/sp1-plonk/cluster-20260813T063649Z-8cpu.md`
Retained proof	Benchmark	`docs/benchmarks/sp1-plonk/cluster-20260825T182353Z-18591.md`
Isolated verifier	Benchmark	`docs/benchmarks/sp1-verifier/verifier-20260825T192214Z-34749.md`
Complete services	Benchmark	Service-flow experiment, 26 August 2026.
Live portal	Functional	Portal enforcement results, 30 August 2026.

The eight-CPU transcript records a source-manifest identity but no commit or image digest. Reproduction requires the pinned toolchain, approved guest, public setup material, and versioned runbooks. We excluded credentials, private snapshots, witness traces, and full capability payloads from this report.

A.3 Supplementary Workflow Artifact

We retained a 1920-by-1080 screen recording as supplementary project material. The file, `Raziel-Final-Demo.mp4`, lasts 22:27.012. Its container metadata records 18 October 2026. SHA-256:

```
ad67b22c3906d99f8699a5361b6069bd2ef00efe2cbdca9cffd19dbdda1a56e3
```

Chapter 6 presents the reconciliation workflow and its outcome directly. The experiment records above supply the timing and resource measurements reported in Chapter 7.