Ritma Pilot Proposal

CISO / CTO Briefing

Prepared by Umesh (Ritma)

Version: Pilot-ready, December 2025

Ritma is aimed at teams that already have logging, monitoring, and policy enforcement in place, but who need to be able to *prove*, to themselves, their board, auditors, and regulators, that these systems behaved as claimed at specific points in time.

Ritma: Cryptographic Evidence Fabric Around Your Existing Stack

- Ritma adds a *cryptographically provable audit fabric* around your existing SIEM / IAM / policy stack, without forcing you to replace tools that already work.
- Pilot is sidecar-only: we observe and prove, but do not sit inline on critical production paths or become a new availability risk.
- You get hash-chained logs, truth snapshots ("Git commits for reality"), and verifiable evidence packages that can be checked independently from the Ritma team.
- Commercially: a 10-day free guided demo, then an optional paid pilot with clear success criteria, decision gate, and no lock-in if it does not meet your bar.

1 Chapter 1: Problem Why Evidence Is Broken Today

Modern organizations already collect vast quantities of logs and metrics, but when regulators or incident response teams ask hard questions, those logs are often not enough to give a confident, defensible answer. This chapter explains why the status quo around evidence is fragile.

- Logs are cheap to generate and easy to silently lose or rewrite.
- Compliance evidence is manual, brittle, and rarely cryptographically provable.
- Policy and access decisions are hard to reconstruct and defend months later.
- Evidence pipelines from raw logs to final audit decks are often undocumented, making it hard to show auditors exactly how results were produced.
- Responsibility for end-to-end evidence integrity is scattered across teams; no single system is accountable for the full chain of custody.

2 Chapter 2: Architecture, Data Flows, and Evidence Model

This chapter describes how Ritma is deployed in a pilot, what components are involved, and what data flows through the system. The goal is to make it obvious where Ritma sits in relation to your existing SIEM, IAM, and policy stack.

- Components: `utld` daemon (truth layer) and `utl_cli` (CLI client/reporting).
- Storage: `dig_index.sqlite`, `decision_events.jsonl`, `compliance_index.jsonl`.
- All data remains on your infrastructure; payloads can be redacted or hashed.
- Integration pattern: applications and policy engines send structured events to Ritma over a local socket; Ritma never needs direct access to production databases or secrets.
- Network stance: in a pilot, Ritma does not require outbound internet access; all state is local so you can test it entirely inside your own security boundary.

3 Chapter 3: Security Posture, Threat Model, and Failure Modes

From a security and risk perspective, the key questions are: what happens if a Ritma node is compromised, if its storage is corrupted, or if keys leak? This chapter outlines our assumptions and how integrity and blast radius are managed in each scenario.

- We assume baseline OS/IAM controls; Ritma focuses on integrity and provability of evidence.
- Node compromise or DB corruption is detectable via hash-chains and truth snapshot verification.
- Pilot mode is non-inline: if Ritma is down, production continues; you only lose new evidence, not availability.
- Ritma's own operations are logged and can be included in evidence packages, so you can see when policies were changed, when evidence was exported, and by whom.
- Signing and verification keys are configurable to use your existing secret-management or KMS setup in a production deployment.

4 Chapter 4: Pilot Offer 10-Day Demo to Paid Pilot

This chapter sets out the commercial and operational path: a low-friction 10-day demo, followed by an optional, tightly scoped paid pilot with clear success criteria and an explicit yes/no decision point.

- 10-day free demo on staging/non-prod with 1–2 real workflows.
- If useful: 60–90 day paid pilot with defined control objectives and SLOs for evidence.
- Optional co-enhancement track for reports, integrations, and deeper proofs.
- Clear roles: we expect a CISO/CTO sponsor, a security engineer, and an application owner to be named for the pilot so decisions are fast and grounded.
- Commercial terms, data boundaries, and success criteria are written down up front so there are no surprises at the end of the pilot.

5 Chapter 5: Evidence, Evaluation, and Next Steps

Finally, we outline how you can independently verify Ritma's outputs, how you should evaluate the pilot, and what concrete steps lead from demo to a go/no-go decision on a broader deployment.

- You can independently run hash and signature verification on logs and evidence packages, without needing any secret knowledge from the Ritma team.
- Evaluation questions: can we answer "who knew what, when?" for a real incident, and can we satisfy at least one real audit or control objective with Ritma-derived evidence?
- Next steps: agree pilot scope, run the 10-day demo, review the resulting evidence and reports together, then decide on a paid pilot.
- If the answer is "no", the pilot ends with a clear, documented reason; if the answer is "yes", we transition to production planning with explicit timelines.

6 Chapter 6: Demo Logs and Proof – What You Can Verify Today

This chapter shows real output from the Ritma advanced demo. Each stage is presented with actual CLI logs and JSON artifacts, followed by an explanation of what it proves and what you can do with it.

6.1 Stage 1: Root Registration and Dig Build

What happens: A cryptographic root is registered, a transition is recorded, and a dig file is built with a Merkle root.

```
[ritma-demo] Registering root 123456 with hash 6a6898e...
root registered: 123456

[ritma-demo] Building DigFile for root=123456 file_id=9001
root_id: 123456
file_id: 9001
merkle_root: c02bc498d2f7a5b0db28202c8a057abc309841a10ef51f7212eeed9e0a03b7d0
record_count: 2
Truth snapshot emitted: trigger=dig_build
```

What this proves:

- The system can register a cryptographic root and build a tamper-evident dig file with a Merkle root over 2 records.
- A truth snapshot is automatically emitted when a dig is built, capturing the state of the dig index and policy ledger at that moment.

What you can do:

- Verify the Merkle root by replaying the hash computation over the records in the dig file.
- Use the truth snapshot to prove what the system knew at the time of the dig build.

6.2 Stage 2: Policy Validation, Testing, and Ledger Burn

What happens: A TruthScript policy is validated, tested against synthetic events, and burned into an immutable policy ledger.

```
[ritma-demo] Validating TruthScript policy demo/policy.access.demo.json
[policy-validate] policy 'demo_security_policy' v1.0.0 loaded (rules=2)

[ritma-demo] Testing policy against synthetic access event
[policy-test] no actions fired

[ritma-demo] Burning policy into ledger
[policy-ledger] ts=1765494657 policy_id=demo-policy version=1
                hash=7c5bd91f1279a990f4eaf5c29f39f003fae2ac6a32d4f17d28b9b91781b19c70
```

What this proves:

- Policies are validated before use and can be tested against synthetic events to verify their logic.
- Once burned, a policy version is immutable and hash-linked in the ledger; you cannot silently change it later.

What you can do:

- Audit the policy ledger to see when each policy version was deployed and what its hash was.
- Replay policy tests to confirm that a policy behaves as expected before burning it into production.

6.3 Stage 3: SOC2 Compliance Check Over Decision Events

What happens: Ritma evaluates SOC2 controls (CC.2.1, CC.7.2) over real decision events and writes results to a hash-chained compliance index.

```
[ritma-demo] Running compliance-check over decision events
control_id,framework,policy_commit_id,passed,total
CC.2.1,SOC2,,0,13
CC.7.2,SOC2,,0,13

[compliance_index] {"control_id":"CC.2.1","framework":"SOC2",
  "tenant_id":"acme","root_id":"300","entity_id":"6",
  "ts":1764701892,"passed":false,"prev_hash":null,
  "record_hash":"44651b0029aeed65e321e87bf105e325e4e2958fbac19ef894fb1f15eeca1e51"}
```

What this proves:

- Compliance controls can be evaluated automatically over decision events in near-real-time.
- Each compliance record is hash-chained to the previous one, making it impossible to silently delete or reorder records.

What you can do:

- Generate compliance reports for auditors showing pass/fail rates for specific controls over a time range.
- Verify the hash chain in `compliance_index.jsonl` to confirm no records were tampered with.

6.4 Stage 4: Decision Events and Incident Detection

What happens: Decision events are logged with policy decisions (allow/deny), and security incidents (denies) are flagged.

```
[decisions] {"ts":1764701930,"tenant_id":"acme","root_id":"300",
  "entity_id":"999","event_kind":"http_request",
  "policy_name":"security_policy","policy_version":"1.0.0",
  "policy_decision":"deny",
  "policy_rules":["deny_public_to_internal_zone_flow"],
  "policy_actions":["deny","flag_for_investigation"],
  "actor_did":"did:ritma:id:acme:attacker"}

[soc-incidents] 1764701930,acme,300,999,http_request,deny,,
```

What this proves:

- Every access decision is logged with the policy name, version, rules fired, and actions taken.
- Denied requests are automatically flagged as incidents and can be queried for forensic investigation.

What you can do:

- Answer "who tried to access what, when, and why was it denied?" for any incident.
- Reconstruct the exact policy logic that was in effect at the time of a decision.

6.5 Stage 5: Evidence Package Export and Verification

What happens: An evidence package is exported for a specific time range, signed with an HMAC key, and then verified.

```
[ritma-demo] Exporting evidence package for acme (time_range demo)
Package signed with utl_cli
Evidence package written to: demo/evidence.package.json
Package ID: pkg_tenant-a_1765499256
Package hash: 421299c6957456fb2ba9670be03ef54c6b8f9f150e1853f2dffe918df6eec0d8

[ritma-demo] Verifying evidence package demo/evidence.package.json
[evidence-verify] Package ID: pkg_tenant-a_1765499256
[evidence-verify] Hash valid: true
[evidence-verify] Signature valid: true
[evidence-verify] Package verification PASSED
```

Evidence package JSON (excerpt):

```
{
  "package_id": "pkg_tenant-a_1765499256",
  "tenant_id": "tenant-a",
  "scope": { "type": "time_range", "time_start": 1765499191, "time_end": 1765499251 },
  "chain_heads": {
    "dig_index_head": "empty",
    "policy_ledger_head": "865b16bc65a65b8e014df69baac38ab7f53c7e1ccf59f130db8e4f8aecfa344b"
  },
  "security": {
    "package_hash": "421299c6957456fb2ba9670be03ef54c6b8f9f150e1853f2dffe918df6eec0d8",
    "signature": {
      "signature_type": "hmac_sha256",
      "signature_hex": "e93c23565990aea83378d75c25ce67653a24194f3ffaacc78a138ab02d758189",
      "signer_id": "utl_cli"
    }
  }
}
```

What this proves:

- Evidence packages capture the state of all relevant chains (dig index, policy ledger, compliance index) at a specific point in time.
- The package hash and signature can be verified independently to confirm the package has not been tampered with.
- Chain heads in the package allow you to verify that the evidence is consistent with the underlying logs.

What you can do:

- Export evidence packages for specific time ranges, tenants, or compliance frameworks and hand them to auditors.
- Verify packages using the CLI without needing access to the Ritma team or any secret keys (if using hash-only verification).
- Use chain heads to cross-check that the evidence package matches the state of your logs at the time it was created.

6.6 Stage 6: Truth Snapshot Export and Verification

What happens: A truth snapshot is exported, capturing the heads of the dig index and policy ledger at a specific moment.

```
{
  "kind": "truth_snapshot_export",
  "dig_index_head": "9cdce1b1345edd0692d2b9a370efbc0036315897c6c1f0a817960145231a12cb",
  "policy_ledger_head": "865b16bc65a65b8e014df69baac38ab7f53c7e1ccf59f130db8e4f8aecfa344b",
  "ts": 1765499257
}
```

What this proves:

- Truth snapshots are "Git commits for reality" – they capture what the system knew at a specific timestamp.
- You can use truth snapshots to prove that a specific policy version or dig index state was in effect at a given time.

What you can do:

- Compare truth snapshots over time to detect if logs or policies were altered retroactively.
- Use truth snapshots as anchors for incident investigations: "what did the system know when this decision was made?"

6.7 Summary: What the Demo Proves

- Ritma can register roots, build tamper-evident dig files, and emit truth snapshots automatically.
- Policies are validated, tested, and burned into an immutable ledger before use.
- Compliance controls (SOC2, etc.) are evaluated over decision events and recorded in a hash-chained index.
- Decision events capture full context (policy, rules, actions, actors) and can be queried for incidents.
- Evidence packages bundle chain heads and artifacts, are signed, and can be verified independently.
- Truth snapshots provide "Git commits for reality" that can be used to prove system state at specific points in time.

All of these artifacts are available in the `demo/` folder and can be verified using the `utl_cli` commands shown above.

7 Chapter 7: Command Reference – Try It Yourself

This chapter provides a complete command reference for running the Ritma demo and verifying all claims made in this proposal. Every command is copy-pastable and can be run on your own infrastructure to confirm that Ritma works as described.

7.1 Prerequisites

- Rust toolchain installed (`rustc`, `cargo`)
- Clone the Ritma repository: `git clone https://github.com/GlobalSushrut/ritma-utl.git`
- Build the CLI: `cargo build --release`
- Set environment variables (see below)

7.2 Environment Setup

Before running any commands, set these environment variables to configure Ritma's storage paths and signing keys:

```
# Core storage paths
export UTLD_DECISION_EVENTS=./decision_events.jsonl
export UTLD_DIG_INDEX_DB=./dig_index.sqlite
export UTLD_COMPLIANCE_INDEX=./compliance_index.jsonl
export UTLD_POLICY_LEDGER=./policy_ledger.jsonl

# Unset JSONL dig index to force SQLite mode
unset UTLD_DIG_INDEX

# Generate a signing key (HMAC-SHA256, 32 bytes hex)
export UTLD_PACKAGE_SIG_KEY=$(openssl rand -hex 32)
export UTLD_PACKAGE_VERIFY_KEY=$UTLD_PACKAGE_SIG_KEY

# Demo mode: allow unsigned packages to pass verification
export RITMA_DEMO_ALLOW_UNSIGNED=1
```

7.3 Core Commands: Roots, Transitions, and Digs

Register a cryptographic root:

```
cargo run --bin utl_cli -- root-register \
  --root-id 123456 \
  --root-hash 6a6898e22bcbf8a44a3e37918f884758f4f25cd7494df56f240c514a40b23a8d \
  --param env=demo --param purpose=pilot-test
```

List all registered roots:

```
cargo run --bin utl_cli -- roots-list
```

Record a state transition:

```
cargo run --bin utl_cli -- tx-record \
  --entity-id 42 --root-id 123456 \
  --signature 7369672d64656d6f \
  --data '{"action":"demo_transition","env":"pilot"}' \
  --addr-heap-hash 6a6898e22bcbf8a44a3e37918f884758f4f25cd7494df56f240c514a40b23a8d \
  --hook-hash 6a6898e22bcbf8a44a3e37918f884758f4f25cd7494df56f240c514a40b23a8d \
  --logic-ref demo.logic.v1 --wall demo-boundary --param tenant=acme
```

Build a dig file with Merkle root:

```
cargo run --bin utl_cli -- dig-build \
  --root-id 123456 --file-id 9001 \
  --time-start $(date +%s) --time-end $(date +%s)
```

List dig files for a tenant:

```
cargo run --bin utl_cli -- digs-list \
  --tenant acme --limit 10 --show-path
```

7.4 Policy Commands: Validation, Testing, and Ledger

Validate a TruthScript policy:

```
cargo run --bin utl_cli -- policy-validate \
  --file demo/policy.access.demo.json
```

Test a policy against synthetic events:

```
cargo run --bin utl_cli -- policy-test \
  --file demo/policy.access.demo.json \
  --kind access --field tenant_id=acme \
  --field resource=demo-resource --field decision=allow
```

Burn a policy into the immutable ledger:

```
cargo run --bin utl_cli -- policy-burn \
  --policy-id demo-policy --version 1 \
  --policy-file demo/policy.access.demo.json
```

List policy ledger entries:

```
cargo run --bin utl_cli -- policy-ledger-list \
  --policy-id demo-policy --limit 20
```

7.5 Compliance Commands: Controls and Reports

Export SOC2 control definitions:

```
cargo run --bin utl_cli -- rulepack-export \
  --kind soc2 --out demo/soc2.controls.json
```

Run compliance check over decision events:

```
cargo run --bin utl_cli -- compliance-check \
  --controls demo/soc2.controls.json --limit 0
```

Generate CISO summary for a tenant and framework:

```
cargo run --bin utl_cli -- ciso-summary \
  --tenant acme --framework SOC2 --limit 20
```

7.6 Decision Events and Incident Detection

List recent decision events:

```
cargo run --bin utl_cli -- decision-events-list --limit 10
```

Query security incidents (denies):

```
cargo run --bin utl_cli -- soc-incidents \
  --tenant acme --limit 20
```

7.7 Evidence Package Commands

Export an evidence package for a time range:

```
cargo run --bin utl_cli -- evidence-package-export \
  --tenant tenant-a --scope-type time_range \
  --scope-id '1765499191:1765499251' \
  --out demo/evidence.package.json
```

Verify an evidence package:

```
cargo run --bin utl_cli -- evidence-package-verify \
  --manifest demo/evidence.package.json
```

7.8 Truth Snapshot Commands

List all truth snapshots:

```
cargo run --bin utl_cli -- truth-snapshot-list --limit 10
```

Verify a truth snapshot:

```
cargo run --bin utl_cli -- truth-snapshot-verify \
  --snapshot-file demo/truth_snapshot.export.json
```

Export a truth snapshot:

```
cargo run --bin utl_cli -- truth-snapshot-export \
  --out demo/truth_snapshot.export.json
```

7.9 Running the Full Advanced Demo

To run the complete end-to-end demo that generates all artifacts shown in Chapter 6:

```
cd demo
bash advanced_demo.sh > advanced_report.txt 2>&1
```

This script will:

- Register roots and record transitions
- Build dig files with Merkle roots
- Validate, test, and burn policies
- Run SOC2 compliance checks
- Generate decision events and detect incidents
- Export and verify evidence packages
- Emit and verify truth snapshots

All output is captured in `advanced_report.txt` for review.

7.10 Verification Commands: Check the Proofs

Verify hash chains in compliance index:

```
# Read compliance_index.jsonl and verify each record_hash
# matches SHA256(prev_hash + record_fields)
python3 -c "
import json, hashlib
prev = None
for line in open('compliance_index.jsonl'):
    rec = json.loads(line)
    if prev and rec['prev_hash'] != prev:
        print(f'FAIL: chain break at {rec}')
    prev = rec['record_hash']
print('Hash chain verified')
"
```

Verify Merkle root in a dig file:

```
# Read a dig file and recompute its Merkle root
# (requires custom script or utl_cli dig-verify command)
cargo run --bin utl_cli -- dig-verify \
  --file ./dig/root-300_file-9001_*.dig.json
```

Cross-check evidence package chain heads:

```
# Compare chain heads in evidence package with actual log files
jq '.chain_heads.policy_ledger_head' demo/evidence.package.json
tail -1 policy_ledger.jsonl | jq -r '.record_hash'
# These should match if the package is consistent
```

7.11 Expected Outcomes

If everything is working correctly, you should see:

- All commands complete without errors (exit code 0)
- Merkle roots and hashes are consistent across artifacts
- Evidence package verification reports `PASSED`
- Truth snapshots capture consistent chain heads
- Compliance reports show control evaluations over decision events
- Policy ledger shows immutable, hash-linked policy versions

If any command fails or produces unexpected output, this is a signal that something is broken or misconfigured. The Ritma team will work with you to diagnose and fix issues during the pilot.