

PrismManifest - README

PrismManifest

Zero-trust tool-argument gate for deterministic AI tool execution.

(Formerly ParamGate - same design; package prismmanifest, CLI prismmanifest-gate.)

License Python Status

PrismManifest sits between probabilistic extractors (LLMs, OCR, table parsers) and deterministic Group 3 compute DAGs. Unverified money values never enter the DAG. Only an Ed25519-signed ParameterManifest that clears the Group 3 boundary is allowed through.

Deterministic Engine + Unverified Probabilistic Input = Deterministic Wrong Answer

PyPI name prismmanifest
Version 0.3.0
Python >= 3.10
License Apache-2.0
Docs (Markdown) docs/
Docs (PDF) docs/pdf/
Usage guide docs/USAGE_FOR_ENGINEERS_AND_ARCHITECTS.

Install (PyPI / pip)

```
pip install prismmanifest
```

With optional extras:

```
pip install "prismmanifest[dev]"
pip install "prismmanifest[cuda]"          # NVIDIA GPU + Numba
pip install "prismmanifest[kms-azure]"     # Azure Key Vault envelope keys
pip install "prismmanifest[kms-aws]"       # AWS KMS envelope (optional)
```

From source (editable):

```
git clone https://github.com/insightitsGit/PrismManifest.git
cd PrismManifest
python -m pip install -e ".[dev]"
python -m pytest -q
```

CLI after install: prismmanifest-gate.

Extra Purpose
dev pytest, coverage, grpcio-tools
docs markdown + fpdf2 (PDF pack builder)
cuda Numba + CUDA 12 wheels (GPU required)
kms-azure / kms Azure Key Vault wrap (preferred cloud KM)
kms-aws boto3 AWS KMS-envelope

How an AI architect should use this

1. Treat LLMs / OCR / parsers as untrusted.
2. Put PrismManifest after extraction and before any calculator, underwriting, or ledger tool that consumes dollar amounts.
3. Route outcomes: ACCEPT -> run DAG - ACCEPT_PENDING_HUMAN -> review - REJECT -> stop.
4. Never let model-generated money text be the tool argument - only evidence-bound, signed manifests.

LLM / OCR -> PrismManifest -> signed ParameterManifest -> Group 3 DAG

Extra LLM benches are optional evidence. They do not change the core gate design. Finding real customer documents matters more for production claims than multi-model FA studies.

Full write-up: Usage for Engineers & AI Architects (and PDF: docs/pdf/01_USAGE_FOR_ENGINEERS_AND_ARCHITECTS.pdf).

How an engineer should integrate this

Minimal end-to-end

```
from prismmanifest import KeyRing, PrismManifestPipeline, enforce_group3_boundary, GateDecision
from prismmanifest.router import DocumentPackage, IntentRouter
from prismmanifest.integrations import demo_capital_gains_dag

keyring = KeyRing.generate(key_id="local-dev-ed25519")

package = DocumentPackage(
    doc_id="1040.txt",
    pages=[
        "Form 1040 Tax Year 2024\n"
        "Line 1 Gross income: $470,000.00\n"
        "Line 11 Adjusted gross income: $450,000.00\n"
    ],
    form_type="IRS_FORM_1040",
    tax_year=2024,
)

# Route -> ingest + span extraction (Pattern B pointers)
routed = IntentRouter().run(package)

# Verify, decide gate, sign manifest
pipeline = PrismManifestPipeline(keyring)
result = pipeline.run_on_evidence(
    evidence=routed.evidence,
    extraction=routed.extraction,
)

# Group 3 hard boundary - this is the security gate
gate = enforce_group3_boundary(
    result.manifest,
    public_keys=keyring,
    expected_dag_id="capital_gains_v3",
)
if gate.decision is GateDecision.ACCEPT:
    receipt = demo_capital_gains_dag(gate.manifest)
    print(receipt)
else:
    print(gate.decision, gate.message)
```

Human escalation

```
from prismmanifest.audit import EscalationQueue
from prismmanifest import PrismManifestPipeline, KeyRing

keyring = KeyRing.generate()
queue = EscalationQueue(".escalation")
pipeline = PrismManifestPipeline(keyring, escalation_queue=queue)
# PASS_WITH_HUMAN manifests are signed and auto-enqueued when a queue is attached.
```

Decorator (DX only - not the security boundary)

```
from prismmanifest import parameter_gated, KeyRing

keyring = KeyRing.load(".keys")

@parameter_gated(public_keys=keyring, expected_dag_id="capital_gains_v3")
def run_dag(*, manifest):
    return manifest.fields[0].value_fixed_micro
```

Production trust must still go through enforce_group3_boundary / gRPC / C++ / in-process prismmanifest_c.

FlatBuffer wire format

```
from prismmanifest.binary_codec import encode_manifest
from prismmanifest.gate import enforce_group3_boundary

buf = encode_manifest(signed_manifest)
result = enforce_group3_boundary(buf, public_keys=keyring, expected_dag_id="capital_gains_v3")
```

Schema: schemas/prismmanifest.fbs.

Do not

- * Skip enforce_group3_boundary because the pipeline "looked good"
- * Feed LLM-printed \$ strings straight into the DAG
- * Treat @parameter_gated alone as the boundary
- * Market cuda_sim / SKIP as CUDA-validated

What it does

1. Ingest evidence with dual-OCR consensus (PDF text + layout re-tokenizer; optional Tesseract).
 2. Ground claims to verbatim spans and form anchors (no generative money values).
 3. Decide PASS / PASS_WITH_HUMAN / REFUSE via quorum, OCR floor (≥ 0.98), and plausibility.
 4. Sign a ParameterManifest (Ed25519) and optionally escalate human review.
 5. Enforce the Group 3 boundary before any DAG runs - Python, gRPC, C++ FlatBuffer, or in-process DLL.
- FinancePackBench and FinancePackBench-G4 provide synthetic SLA / adversarial suites.

Gate model

GateStatus (on the manifest)

Status Meaning
PASS Span-grounded, plausibility OK, no disag
PASS_WITH_HUMAN Immaterial disagreement ($\leq \$1k$), low OC

REFUSE | Not grounded, plausibility failure, or m

GateDecision (Group 3 boundary)

Decision When
ACCEPT PASS, or cleared PASS_WITH_HUMAN with va
ACCEPT_PENDING_HUMAN PASS_WITH_HUMAN awaiting review
REJECT REFUSE, bad/missing clearance, or attest

Attestation, freshness (signed_at_unix skew, default 300s), and replay (ReplayGuard) failures raise GateError.

CLI

```
# Keys
prismmanifest-gate gen-keys --out .keys --key-id local-dev-ed25519
prismmanifest-gate gen-hsm-key --out .hsm --key-id prod-ed25519
prismmanifest-gate gen-kms-key --out .kms --mode local --key-id kms-dev
# Azure: prismmanifest-gate gen-kms-key --out .kms --mode azure --kms-key-id <vault-key-url>

# Sign / verify
prismmanifest-gate sign --keys .keys --manifest manifest.json --out signed.json
prismmanifest-gate sign --keys .keys --manifest manifest.json --out signed.fbs --flatbuffer
prismmanifest-gate verify --keys .keys --manifest signed.json --dag-id capital_gains_v3

# Benches & proof
prismmanifest-gate bench --packages 500
prismmanifest-gate bench --require-cuda --packages 40
prismmanifest-gate bench-manifest-parity --packages 500 --require-cuda
prismmanifest-gate bench-perf --out reports/perf
prismmanifest-gate bench-customer-pdf --corpus .corpus/customer --seed-synthetic
prismmanifest-gate compliance --out reports/compliance
prismmanifest-gate pilot-pack --out reports/pilot_pack
prismmanifest-gate g4-suite --out reports/g4 --fuzz 200

# Ops / review
prismmanifest-gate audit-replay --store .audit --receipt <id> --keys .keys
prismmanifest-gate escalation-list --queue .escalation
prismmanifest-gate review-ui --queue .escalation --keys .keys --bind 127.0.0.1:8766
```

gRPC Group 3 service

Proto: proto/prismmanifest_gate.proto Service: prismmanifest.v1.Group3Gate - VerifyManifest, ExecuteDag

```
from prismmanifest.attestation import KeyRing
from prismmanifest.grpc_servicer import serve

serve(KeyRing.load(".keys"), bind="[:,]:50051")
```

C++ / in-process gate (optional)

```
# Windows
powershell -File scripts/build_cpp_gate.ps1
# Produces: prismmanifest_gate_enforce.exe + prismmanifest_c.dll

prismmanifest_gate_enforce signed.fbs public.pem <key_id> capital_gains_v3
```

Python can call the shared library without process spawn:

```
from prismmanifest.cpp_bridge import enforce_fb, bridge_status
print(bridge_status()) # inprocess_available when PRISMMANIFEST_C_DLL / build present
```

Ops packaging (pilot deploy)

"Ops packaging" = how you run the gate in a service (not the algorithm):

- * Keys (local / software-HSM / Azure KV)
- * RBAC, timeouts, idempotency, metrics (prismmanifest.ops)
- * Audit store + human review UI
- * Optional C++ / CUDA beside the Python package

See docs/handoff/PILOT_DEPLOY.md.

Documentation (Markdown + PDF)

Doc Role
USAGE_FOR_ENGINEERS_AND_ARCHITECTS.md How to use PrismManifest
PRISMMANIFEST_SYSTEM_DESIGN.md Architecture & threat model (authority)
PRISMMANIFEST_IMPLEMENTATION_PLAN.md Phases & SLAs
PRISMMANIFEST_MASTER_SPECIFICATION.md Executive index
FINANCEPACKBENCH_G4_ADVERSARIAL_SUITE.md Adversarial suite
FINANCEPACKBENCH_PROMPT_INJECTION_SEMANT Why inert injection can still PASS
docs/pdf/ Generated PDF pack (including full combi

Regenerate PDFs:

```
pip install "prismmanifest[docs]" # or: pip install markdown fpdf2
python scripts/build_docs_pdf.py
```

Layout

prismmanifest/	Python package
cpp/gate/	C++ canonicalize + FlatBuffer enforce + prismmanifest_c
cuda/kernels/	Experimental .cu kernels
schemas/	prismmanifest.fbs
proto/	gRPC Group3Gate
tests/	pytest
docs/	Specs + usage + PDF output
scripts/	proto, C++ build, CUDA shim, PDF builder
reports/	Generated proof artifacts (not required for pip install)

Honest scope & readiness

Label Meaning
Pilot OSS Synthetic + adversarial FA=0 under test;
Production claim Requires your live customer fax/scanned

Also:

- * Security boundary = enforce_group3_boundary / gRPC / C++ / prismmanifest_c - not @parameter_gated alone.
 - * CUDA: real parity needs GPU + Numba; cuda_sim is CI self-check only.
 - * HSM: gen-hsm-key is software encrypted-at-rest, not PKCS#11 hardware.
 - * Prompt injection: PrismManifest is an execution trust gate - see the injection semantics doc before claiming "injection defense."
- Publishing notes: docs/PUBLISHING.md.
-

License

Apache License 2.0 - see LICENSE.