DigiByte Quantum Shield Network – Code Blueprint

DigiByte Quantum Shield Network – Code Blueprint (v1)

Repository layout (Python prototype):

- LICENSE - README.md - dqsn_engine.py   # Core quantum risk engine - dqsn_core.py   # FastAPI JSON API exposing the engine

1. dqsn_engine.py – Core Module

Key components:

- Utility functions   - _byte_histogram(data: bytes) -> List[int]   - shannon_entropy(data: bytes) -> float   - repetition_ratio(data: bytes) -> float

- Data models   - @dataclass QuantumRiskInput   sig_entropy: float   sig_repetition: float   mempool_spike: float   reorg_depth: int   cross_chain_alerts: int

  - @dataclass QuantumRiskResult   risk_score: float   level: str   factors: Dict[str, float]

- Normalisation helpers   - _normalize_entropy(ent: float) -> float   - _normalize_repetition(r: float) -> float   - _normalize_reorg(depth: int) -> float   - _normalize_alerts(count: int) -> float

- Main API   - compute_risk(input: QuantumRiskInput) -> QuantumRiskResult   - classify_level(risk_score: float) -> str   - analyze_signature(signature_bytes: bytes, mempool_spike=0.0, reorg_depth=0, cross_chain_alerts=0) -> QuantumRiskResult

- Demo entry point   - _demo_signatures() – compares “good” vs “suspicious” signatures when module is executed as __main__.

2. dqsn_core.py – FastAPI Shim

This module is a simple JSON API wrapper around the engine. It is intentionally small so that DigiByte developers can replace FastAPI with any other framework if desired.

Main elements:

- Pydantic models: - BlockMetrics – high■level view of local chain metrics such as entropy_bits_per_byte, nonce_reuse_rate, mempool_utilization, etc.   - AnalyzeRequest – input payload with metrics, source_chain and optional window_label.   - RiskAssessment and AnalyzeResponse – output structures with risk_score, level, recommended_action and details.

- Endpoint: - POST /dqsnet/analyze   Input: AnalyzeRequest (JSON)   Output: AnalyzeResponse (JSON)

This endpoint internally calls compute_risk_score(...) and returns a structured assessment. In a future version, the implementation can be refactored to call dqsn_engine.compute_risk(...) directly.

3. Integration Points

- Node plugins: a DigiByte node can collect local metrics (mempool size, reorg depth, taproot adoption) and forward them periodically to /dqsnet/analyze. - Sentinel AI: can treat dqsn_engine.analyze_signature(...) as a library call inside its own detection pipeline. - External chains: may submit their own byte sequences and context to reuse DQSNet as a general quantum firewall engine.

4. Testing

The engine module includes a built■in demo. Developers can extend this with proper unit tests by asserting:

- Entropy and repetition behave as expected for random vs low■entropy data. - Risk scores rise as mempool_spike, reorg_depth, or cross_chain_alerts increase. - Level mapping boundaries (0.24/0.25/0.49/0.50 etc.) remain correct.