DigiByte Quantum Shield Network – Whitepaper

DigiByte Quantum Shield Network (DQSNet) – Whitepaper (v1)

Author: DarekDGB License: MIT

1. Vision

The DigiByte Quantum Shield Network (DQSNet) is a community■driven security layer designed to protect DigiByte and other UTXO chains from emerging quantum threats. It does not replace consensus. Instead, it adds a specialised “early warning and response” layer that can be attached to full nodes, exchanges, and custodians.

The core idea is simple: measure the health of cryptographic randomness and chain stability in real time, turn those signals into a clear risk score, and provide automated hooks that can trigger quantum■safe responses when thresholds are crossed.

2. Threat Model

DQSNet focuses on detecting early signs of large■scale key extraction or transaction forgery enabled by powerful quantum adversaries using algorithms such as Shor’s or Grover’s. The project assumes:

- Existing DigiByte signatures (ECDSA/secp256k1) may become vulnerable in the future. - Attackers will likely probe the network gradually, reusing nonces or signatures, or pushing abnormal transaction patterns. - High■value targets (exchanges, custodians, bridges) need actionable alerts before a catastrophic exploit occurs.

The network does not try to solve every quantum problem. Instead, it aims to be an honest, explainable watchdog that can be wired into post■quantum migration plans.

3. Architecture Overview

DQSNet is split into two main layers:

1) dqsn_engine.py – Core risk engine - Pure■Python, self■contained module. - Computes Shannon entropy and repetition ratios for signature■like byte sequences. - Combines local signature statistics with network context (mempool spikes, reorg depth, cross■chain alerts) into a single risk score between 0.0 and 1.0. - Classifies the score into four shield levels: normal, elevated, high, critical.

2) dqsn_core.py – API and integration shim - FastAPI■based HTTP service that exposes the risk engine over a simple JSON API. - Designed to be called by DigiByte nodes, Sentinel■AI agents, monitoring dashboards, or external chains. - Returns structured risk assessments with factors and recommended actions.

This separation keeps the engine small and embeddable while allowing multiple API layers (HTTP, gRPC, RPC, CLI) to grow around it over time.

4. Risk Scoring Model

The v1 engine uses an intentionally transparent weighted model. Its inputs are:

- sig_entropy: Shannon entropy of the signature bytes (0.0–8.0 bits/byte). - sig_repetition:

Ratio of repeated bytes to total bytes (0.0–1.0). - mempool_spike: Normalised indicator of current mempool stress (0.0–1.0). - reorg_depth: Observed depth of the latest chain reorganisation. - cross_chain_alerts: Count of external alerts from other chains or security feeds.

Each input is normalised into a 0.0–1.0 risk factor. For example, low entropy is mapped to high risk, while higher entropy maps to low risk. Factors are then aggregated with tunable weights:

- entropy: 0.30 - repetition: 0.25 - mempool: 0.15 - reorg: 0.15 - alerts: 0.15

The resulting score in [0, 1] is mapped to a clear shield level:

- 0.00–0.24 → normal - 0.25–0.49 → elevated - 0.50–0.74 → high - 0.75–1.00 → critical

5. Intended Responses

DQSNet does not enforce policy itself; instead it provides data and suggested actions. Example responses per level:

- normal: routine monitoring only. - elevated: archive metrics, notify operators, and accelerate PQC testing. - high: rotate keys where possible, lower withdrawal limits, and increase logging. - critical: trigger emergency PQC runbooks, pause high■risk flows, and coordinate with DigiByte core and major exchanges.

6. Roadmap

Planned future directions include:

- Feeding DQSNet with live DigiByte node metrics and mempool snapshots. - Extending the engine with time■series features and machine learning. - Adding hooks for post■quantum signature schemes (e.g., Dilithium) and hybrid address formats. - Publishing reference integrations for exchanges, wallets, and Sentinel■AI deployments.

DQSNet is intentionally MIT■licensed so that any project in the wider ecosystem can reuse, modify, and build upon the code without restriction, while giving attribution to DarekDGB and contributors.