SoftwareX 34 (2026) 102676

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original Software Publication

QRNG-DD: software data diode for quantum random number distribution with AI agent integration

Valer Bocan

Department of Computer and Information Technology, Politehnica University of Timisoara, Timisoara 300223, Romania

ARTICLE INFO

Keywords:
Quantum random number generator
Data diode
Entropy distribution
Model context protocol
Rust
Network security

ABSTRACT

QRNG-DD is an open-source research infrastructure for securely distributing quantum random numbers across network boundaries. The system implements a software-based data diode architecture that allows researchers to access quantum randomness from protected internal networks while maintaining strict security isolation. We designed QRNG-DD to support quantum computing experiments, cryptographic studies, and the emerging paradigm of AI-assisted research workflows.

Our Rust implementation achieves high-performance entropy delivery with throughput limited by QRNG hardware rather than software overhead. Multi-source entropy aggregation using XOR or HKDF (HMAC-based Key Derivation Function) mixing defends against single-vendor failures and potential backdoors. The Model Context Protocol (MCP) integration allows AI agents to consume quantum randomness through standardized tools, supporting autonomous scientific workflows in quantum computing, machine learning, and computational physics.

The system addresses key research challenges: transparent entropy distribution for reproducible studies, AI-accessible quantum randomness for autonomous research agents, high-throughput delivery for Monte Carlo simulations, and affordable deployment for academic institutions. All source code and benchmark artifacts are available under MIT license.

Metadata

Nr	Code metadata description	*Metadata*
C1	Current code version	v1.0.0
C2	Permanent link to code/repository used for this code version	https://github.com/vbocan/qrng-data-diode
C3	Permanent link to reproducible capsule	
C4	Legal code license	MIT License
C5	Code versioning system used	Git
C6	Software code languages, tools and services used	Rust 1.75+
C7	Compilation requirements, operating environments and dependencies	Rust 1.75+ toolchain (Rustls, no OpenSSL), tokio 1.48, axum 0.8, hmac 0.12, sha2 0.10; Linux, macOS, Windows; Docker optional
C8	If available, link to developer documentation/manual	https://github.com/vbocan/qrng-data-diode/blob/master/README.md
C9	Support email for questions	valer.bocan@upt.ro

1. Motivation and significance

Quantum Random Number Generators (QRNGs) derive randomness from fundamental quantum processes such as photon detection, vacuum fluctuations, or radioactive decay, offering a qualitatively different security foundation than pseudo-random number generators [1–4]. While algorithmic generators produce deterministic sequences from initial seeds, QRNGs exploit the inherent unpredictability of quantum mechanics to generate truly random bits. This distinction carries practical weight for cryptographic key generation [5], scientific simulations requiring unbiased sampling [6], and security protocols where predictability constitutes vulnerability [7].

Commercial QRNG appliances such as ID Quantique's Quantis are frequently deployed on isolated internal networks due to organizational security policies [8]. Network segmentation protects these devices from external threats but creates a paradox: quantum entropy is protected yet inaccessible to researchers, AI systems, and external applications that need it. The current NISQ (Noisy Intermediate-Scale Quantum) era demands extensive characterization of quantum devices, where

E-mail address: valer.bocan@upt.ro.

https://doi.org/10.1016/j.softx.2026.102676
Received 4 January 2026; Received in revised form 7 April 2026; Accepted 20 April 2026
Available online 4 May 2026
2352-7110/© 2026 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

randomized benchmarking protocols consume substantial entropy for generating random Clifford sequences [9]. Researchers face a choice between compromising network security or forgoing quantum randomness entirely.

Hardware data diodes enforce unidirectional data flow through physical constraints but cost $5000 to $50,000 and require specialized infrastructure [10,11]. Fiber-optic implementations physically prevent return signals, offering strong guarantees suitable for critical infrastructure. Such costs are prohibitive for most academic institutions. Public QRNG services from ANU and NIST expose quantum entropy via Internet APIs [12,13]; however, they impose rate limits of 5 requests per second, restrict response sizes to 1 KB, add substantial latency (~450 ms for ANU [12]), and raise privacy concerns through centralized logging. For researchers running Monte Carlo simulations requiring millions of random samples, these constraints render public services impractical.

The emerging field of AI-assisted research compounds these challenges. Autonomous agents increasingly support scientific workflows, from experimental design to hypothesis iteration [14]. Large language models now orchestrate complex research tasks, yet must access QRNG services through ad hoc HTTP clients rather than standardized tools. The absence of native AI integration forces researchers to build custom adapters, duplicating effort across institutions.

We developed QRNG-DD to bridge this gap. The system implements a software-based data diode architecture that distributes quantum entropy without hardware data diode costs. An Entropy Collector on the internal network fetches quantum data from QRNG appliances, signs and encapsulates each packet, then pushes to an Entropy Gateway on the external network. The Gateway never initiates reverse connections, enforcing unidirectional data flow through architectural constraints rather than physical isolation [10]. Firewall rules complement the software design, blocking all inbound traffic to the Collector while permitting outbound pushes.

We implemented QRNG-DD in Rust, leveraging compile-time memory safety guarantees [15,16] and asynchronous I/O through Tokio [17] for high-throughput entropy delivery. The Model Context Protocol integration exposes quantum randomness through standardized JSON-RPC 2.0 tools [18], while multi-source aggregation using XOR or HKDF mixing addresses vendor backdoor concerns through information-theoretic security guarantees [19].

QRNG-DD serves infrastructure needs across several research domains. In quantum computing, researchers use it for state initialization, gate testing, and error correction validation [20,9]. Variational quantum eigensolvers and quantum approximate optimization algorithms require random initial parameters; biased initialization can trap optimization in local minima that mask true algorithm performance. Machine learning applications include weight initialization [21], dropout regularization [22], and data augmentation. Recent work demonstrates that initialization quality affects not just convergence speed but final model accuracy. Cryptographic research benefits from key generation, nonce production, and backdoor-resistant protocol studies. The discovery of exploitable weaknesses in commercial random number generators, notably the Dual_EC_DRBG algorithm, underscores the importance of vendor-independent entropy sources. Computational physics applications span Monte Carlo methods [6,23], random walks, and statistical mechanics simulations. High-precision simulations for lattice QCD and protein folding require entropy sources that pass stringent uniformity tests across billions of samples. The MCP integration particularly supports AI-assisted research, where autonomous agents design experiments and iterate hypotheses [18].

Hardware data diodes from vendors like Owl Cyber Defense deliver physical unidirectional guarantees through fiber-optic transmission [10]. They offer maximum security but require substantial investment that few research institutions can justify. QRNG-DD trades physical guarantees for practical software isolation at significantly reduced cost, suitable for research environments and moderate-security deployments. Public QRNG services from ANU and NIST democratize access but impose limitations unsuitable for intensive research [12,13]: rate limits that throttle Monte Carlo simulations, response size caps that fragment large requests, latency that disrupts interactive workflows, and privacy concerns from centralized logging. Our self-hosted architecture eliminates these constraints while ensuring complete privacy.

Prior work from our department developed libqrng [24], a C library offering direct API access to ID Quantique's Quantis appliances for embedded integration. While libqrng serves local applications effectively, it does not address cross-network distribution, security isolation, or AI integration. QRNG-DD extends this foundation by delivering network-distributed quantum entropy access through a software data diode architecture, implemented in Rust for compile-time memory safety [15,25]. Academic literature on software data diodes remains limited [26].

A small number of community-maintained MCP servers expose randomness to AI agents, but they do so by wrapping host operating-system PRNGs (/dev/urandom, language standard libraries) or by re-exposing existing public services such as random.org. None of these implementations provide certified quantum entropy, none verify packet integrity end-to-end, and none address the network-isolation problem that motivates this work; they treat randomness purely as a convenience feature for tasks such as dice rolling or sample shuffling. To our knowledge, QRNG-DD is the first open-source system combining software-enforced unidirectional data flow, quantum entropy distribution, and AI agent integration through the Model Context Protocol [18].

2. Software description

QRNG-DD implements a three-tier research infrastructure, as illustrated in Fig. 1.

The Entropy Collector operates on the internal network, fetching quantum entropy via HTTPS from QRNG appliances, signing packets with HMAC-SHA256 [27], and pushing data outward. We implemented the Collector as a standalone daemon that maintains persistent connections to configured QRNG sources, polling at configurable intervals (default 100 ms) and aggregating responses before transmission. The Entropy Gateway on the external network verifies signatures, buffers entropy in a 10 MB circular buffer, and serves an authenticated REST API [28]. The buffer size reflects empirical analysis: at 80 KB/s appliance throughput and typical research workload patterns, 10 MB provides sufficient headroom for burst requests while limiting memory footprint. The MCP server exposes quantum randomness to AI agents via JSON-RPC 2.0. Because data flows only from Collector to Gateway with no reverse communication, the architecture enforces unidirectional data flow through software constraints.

This architecture addresses a concrete deployment constraint. In our deployment at Politehnica University of Timisoara, the Quantis QRNG appliance resides on the university's internal network, protected by a firewall that blocks all inbound connections from external networks. The Collector, co-located on this internal network, is the only component with direct access to the appliance. It initiates all outbound connections to the Gateway, which is hosted on an external server accessible to researchers. This design requires no changes to the university's network security policies: the internal firewall permits the Collector's outbound HTTPS pushes while continuing to block all unsolicited inbound traffic. Other institutions can replicate this deployment model by operating a Collector alongside their QRNG appliance on an internal network and hosting the Gateway on a DMZ server or at an external provider. Multiple independent clients connect to the Gateway's REST API using bearer token authentication, enabling shared access to quantum entropy across research groups. The current implementation targets single-institution deployments with optional external sharing; the layered architecture could support multi-tenant subscription models in future extensions.

Configuration via environment variables supports both development and production scenarios. Operators can adjust buffer sizes,

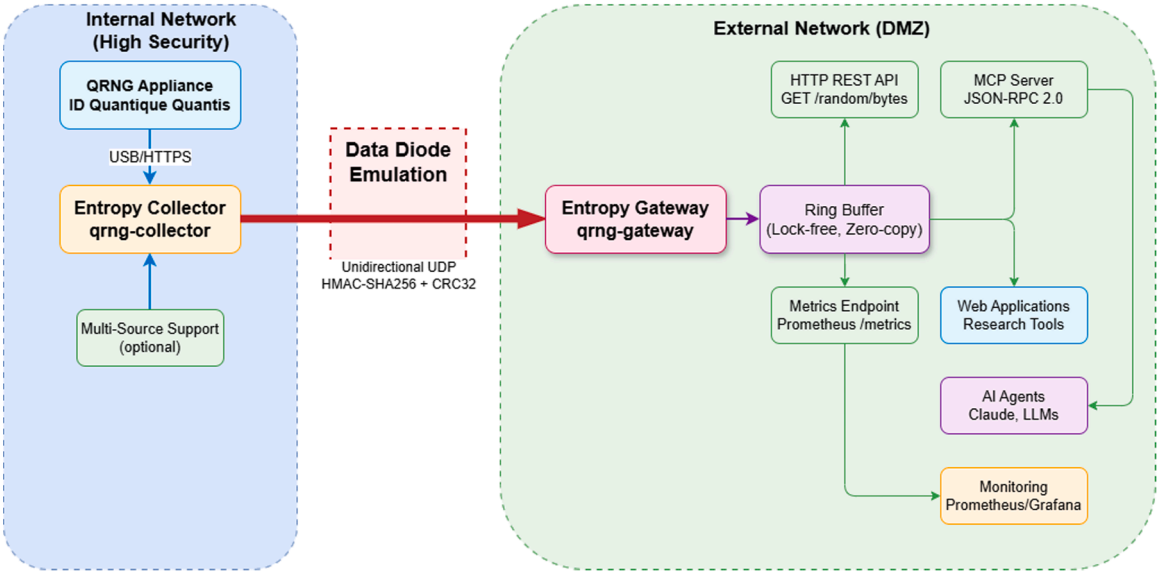

Fig. 1. QRNG-DD Architecture. Data flows unidirectionally from Collector to Gateway. The Gateway never initiates connections to the internal network.

authentication tokens, source endpoints, and logging verbosity without recompilation. Docker containerization ensures reproducible deployment across Linux, macOS, and Windows; the multi-stage Dockerfile produces minimal images under 20 MB. Native binaries run without container overhead when preferred, with single-file deployment simplifying installation on research clusters. Prometheus metrics and structured logging support performance analysis: buffer levels, latency histograms, per-source health tracking, and request counts partitioned by client.

Rust's zero-cost abstractions and asynchronous I/O allow Gateway processing with negligible overhead, making the system suitable for both interactive workflows and Monte Carlo simulations. Sustained throughput depends on QRNG appliance entropy generation rather than software processing; when aggregating from multiple sources, parallel requests achieve aggregate latency approaching the maximum of individual sources.

Four independent integrity layers ensure entropy authenticity, as shown in Fig. 2. HMAC-SHA256 signatures prevent forgery [27], while CRC32 checksums detect transmission errors with high probability. Timestamp validation prevents replay attacks within configurable TTL windows, rejecting packets older than the threshold, and sequence numbers guarantee monotonic ordering to detect dropped or reordered packets. All checks must pass for packet acceptance, creating auditable entropy provenance suitable for peer-reviewed publications. Failed verifications trigger alerts through the metrics system, enabling operators to diagnose network issues or potential attacks.

The MCP server exposes quantum randomness through six standardized JSON-RPC 2.0 tools. Four data generation tools offer: random bytes with configurable encoding (hex, base64, raw) for cryptographic experiments; random integers within specified ranges for algorithm testing; random floats in the unit interval for simulations; and quantum-seeded UUIDs for experimental tracking. Two utility tools complement these: a status endpoint for monitoring Gateway health and buffer levels, and a Monte Carlo π estimation tool for quality validation that samples points and estimates π through the ratio of points inside a unit circle. Each tool returns structured JSON with source attribution marking entropy as quantum-derived. This allows AI agents to cite quantum sources in automated research reports and distinguish from pseudo-random baselines. The interface eliminates custom integration code, reducing AI agent quantum randomness access from hours of HTTP client development to simple tool invocation [18].

Multi-source aggregation addresses vendor backdoor concerns and

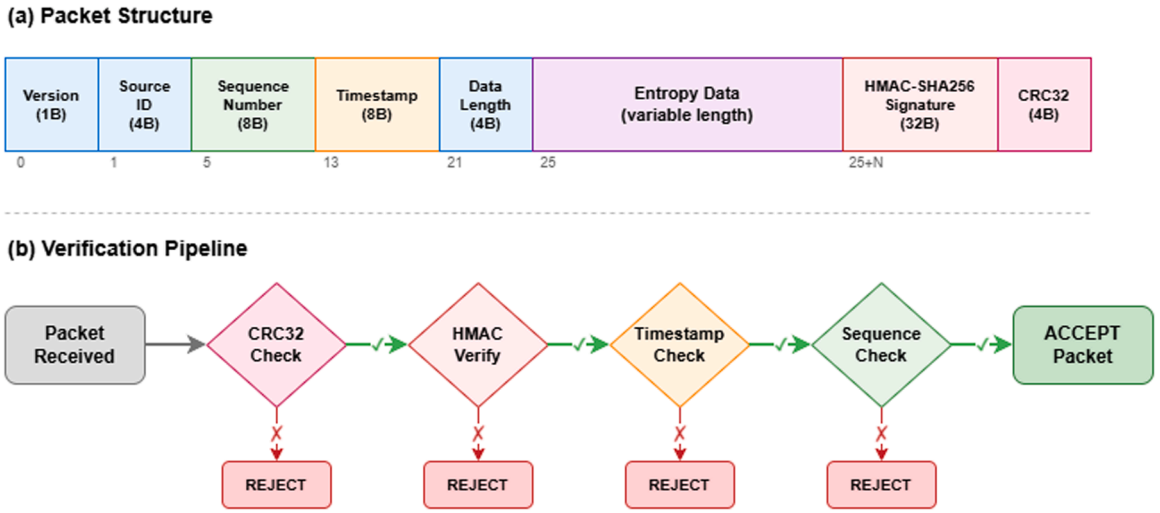

Fig. 2. Entropy packet structure. Each packet carries an HMAC-SHA256 signature, CRC32 checksum, timestamp, and sequence number.

hardware failure risks, as illustrated in Fig. 3. XOR combination offers information-theoretic security: if at least one source generates uniform random bits, the output inherits that uniformity regardless of other sources [19]. This property holds without cryptographic assumptions and even if compromised sources attempt to bias the output. HKDF-based mixing handles potentially correlated sources through cryptographic extraction [29], producing uniform output from non-uniform inputs. The implementation uses HMAC—HA256 as the underlying hash function, following the extract-then-expand paradigm of RFC 5869 [29]. The extract phase computes HMAC-SHA256 over the concatenation of all source entropy chunks, keyed with a deterministic salt derived from the number of active sources. The expand phase uses standard counter-based iteration to produce output matching the input chunk size. While XOR mixing requires source independence, HKDF relaxes this assumption, extracting uniform output even when sources share partial correlations. Automatic health monitoring implements fault isolation with configurable retry intervals, temporarily excluding unresponsive sources while continuing operation. This design supports heterogeneous QRNG arrays resilient to single-point failures.

Automated testing spans unit tests for individual components, integration tests verifying end-to-end flows, and property-based testing for cryptographic functions that generates thousands of random inputs to verify invariants. The property-based tests verify that HMAC signatures remain valid across message transformations, that buffer operations maintain data integrity, and that the mixing functions preserve entropy properties. GitHub Actions executes the test suite on every commit across Linux, macOS, and Windows runners, with coverage reporting tracking tested code paths. Pull requests require passing tests and code review before merging.

We designed QRNG-DD with extensibility as a core principle. Custom entropy sources integrate via Rust trait implementations. Researchers can add support for additional QRNG vendors (Quantis, ComScire, QuintessenceLabs) or alternative entropy sources (hardware security modules, radiation-based RNGs) by implementing the EntropySource trait. The fetcher module handles connection pooling and retry logic automatically. Custom mixing strategies extend the aggregation layer. Beyond the built-in XOR and HKDF strategies, researchers can implement domain-specific mixing algorithms. For example, cryptographic applications might implement entropy pooling with additional whitening, while physics simulations might prefer different combination strategies. Pluggable test suites extend validation beyond Monte Carlo π estimation. The example infrastructure demonstrates integration patterns for NIST SP 800–22 [30] and Dieharder [31]. Researchers can add custom statistical tests relevant to their domain. The shared qrng-core library offers reusable Rust components (buffer management, cryptographic signing, protocol handling) applicable to broader research software development. All extension points are documented with examples in the repository.

3. Illustrative examples

The following example demonstrates AI agent interaction with quantum entropy through the Model Context Protocol:

MCP Request: Fetching quantum random integers

```
{
  "jsonrpc": "2.0",
  "method": "tools/call",
  "params": {
  "name": "get_random_integers",
  "arguments": { "count": 10, "min": 1, "max": 100 }
```

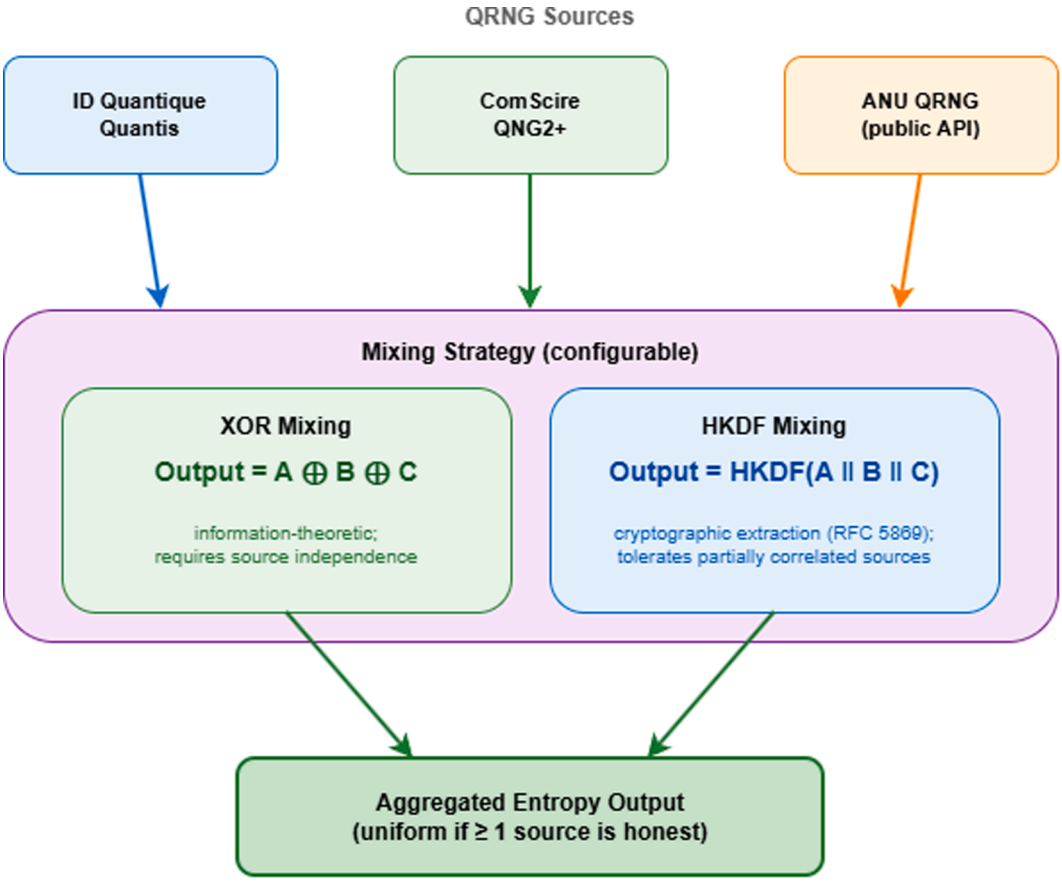

Fig. 3. Multi-source entropy aggregation. Independent QRNG sources are combined through either XOR (information-theoretic) or HKDF (cryptographic extraction) mixing; output uniformity is preserved as long as at least one source is trustworthy.

```
  },
  "id": 1
}
```

MCP Response:

```
{
  "jsonrpc": "2.0",
  "result": {
  "integers": [42, 17, 83, 56, 91, 28, 64, 39, 75, 12],
  "source": "quantum",
  "timestamp": "2025-01-15T10:23:45Z"
  },
  "id": 1
}
```

To illustrate how AI agents leverage quantum entropy in practice, consider the following research workflows in which the agent itself is the entropy consumer rather than a passive conduit. In a Bayesian hyperparameter search, the agent samples each candidate configuration directly from quantum random floats obtained via get_random_floats, dispatches the trial to a training routine, observes the resulting loss, and uses fresh quantum samples to draw the next configuration in its reasoning loop. The randomness governs the agent's own decisions about which region of the search space to explore next; replacing it with a downstream PRNG would relocate the choice away from the agent and break the provenance chain. A second workflow employs stochastic optimization: an agent running a simulated-annealing schedule for combinatorial design uses get_random_floats to draw the Metropolis acceptance threshold at each step, fetches problem-specific samples through get_random_integers, and periodically calls get_data_quality to confirm that the entropy source remains within statistical bounds before continuing. A third workflow targets reproducibility: the agent fetches a seed bundle once at the start of an experiment, records the bundle and its quantum source attribution as part of the experiment metadata, and derives all subsequent sub-streams from this audited root, so the entire run can be replayed and cited by reviewers. In each case the agent discovers the tools through MCP capability negotiation and invokes them as standard function calls within its reasoning loop; the "source": "quantum" attribution returned with each response lets the agent label quantum-derived randomness in automated research reports and distinguish it from pseudo-random baselines. Recent work has highlighted both the value of certified randomness in machine-learning pipelines and the rise of agentic scientific workflows that need standardized entropy access [32,33].

For direct REST API access, applications can request entropy via HTTP:

```
curl -H "Authorization: Bearer $TOKEN" \
https://gateway.example.com/api/v1/entropy?bytes=1024
```

The response contains raw quantum entropy with metadata indicating source and timestamp. Clients can request up to 1MB per request, far exceeding the 1KB limits imposed by public services. The API supports multiple output formats—raw binary, hexadecimal, or Base64—for different application requirements. Authentication uses bearer tokens with configurable expiration, and rate limiting protects against resource exhaustion while ensuring fair access across concurrent clients.

The built-in Monte Carlo π estimation validates statistical quality through geometric sampling:

```
let samples: u64 = 1_000_000;
let inside = (0..samples)
  .filter(|_| {
  let x: f64 = client.get_random_float();
  let y: f64 = client.get_random_float();
  x * x + y * y <= 1.0
  })
  .count();
let pi_estimate = 4.0 * inside as f64 / samples as f64;
```

This consistently produces estimates within 0.01% of π across repeated trials, meeting NIST SP 800-22 criteria [30] for randomness quality. The convergence rate matches theoretical predictions for truly random samples, providing empirical validation that the entropy source exhibits no detectable bias.

Cryptographic applications request entropy directly through the REST API. The following example generates a 256-bit AES key:

```
let entropy = client.get_random_bytes(32)?;
let key = aes_gcm::Key::<Aes256Gcm>::from_slice
(&entropy);
//Key is now ready for authenticated encryption
```

Quantum-derived key material provides security guarantees unavailable from deterministic generators, particularly for long-term secrets.

The repository includes 15 Rust applications demonstrating quantum randomness across research domains: Monte Carlo π estimation with configurable sample counts, genetic algorithms with quantum-random mutation and crossover operators, simulated annealing for combinatorial optimization problems, random walk simulations for diffusion and Brownian motion modeling, cryptographic key generation supporting AES-256 and ChaCha20 algorithms, and statistical test suites for entropy validation. Each example follows idiomatic Rust patterns with comprehensive error handling. PowerShell scripts automate benchmarking and validation pipelines. All examples include source code, documentation, and sample outputs.

4. Impact

QRNG-DD serves as infrastructure for quantum computing experiments requiring genuine quantum randomness. Researchers generate random quantum states for algorithm testing and create input vectors for validating quantum gate implementations. Error correction research benefits from quantum-random error patterns that avoid the subtle correlations present in pseudo-random sequences. The MCP integration allows AI agents to autonomously design experiments, generate test vectors, and analyze results, accelerating research cycles beyond what manual workflows permit. An AI agent can request quantum random parameters, execute quantum circuits, analyze results, and iterate on experimental design without human intervention at each step.

AI-assisted research workflows benefit from standardized quantum entropy access. Neural network training depends on weight initialization, where quantum randomness avoids biases that pseudo-random generators can introduce [21]. Studies demonstrate that poor initialization can increase training time by orders of magnitude or prevent convergence entirely. Stochastic optimization, dropout regularization [22], and data augmentation pipelines all require high-quality randomness. AI agents access quantum entropy conversationally through MCP tools, eliminating integration barriers that previously required custom HTTP client development for each project.

Monte Carlo simulations consume vast quantities of unbiased entropy [6,23]. Poor-quality random number generators can introduce hidden errors in simulation results that propagate through subsequent analysis [34]. Random walk studies, statistical mechanics simulations, and Markov Chain Monte Carlo methods [35] achieve their theoretical convergence properties only with sufficiently high-quality randomness. Ferrenberg, Landau, and Wong demonstrated that flawed random number generators produced systematic errors in Ising model simulations that appeared statistically significant [34]. The built-in Monte Carlo π estimation validates statistical quality, consistently passing NIST SP 800–22 test criteria [36].

Multi-source entropy aggregation makes QRNG-DD valuable for cryptographic studies requiring vendor-independent randomness [7]. The XOR and HKDF mixing strategies demonstrate practical defense against vendor backdoors, offering reproducible infrastructure for security analysis. The Dual_EC_DRBG incident demonstrated that even widely-deployed random number generators can contain exploitable weaknesses; multi-source aggregation provides defense-in-depth against similar vulnerabilities. Researchers studying post-quantum

cryptography require entropy sources free from potential classical vulnerabilities, as quantum computers may eventually compromise algorithms that depend on computational hardness assumptions. The standardized test infrastructure supports randomness quality comparisons using NIST SP 800–22 [36] and Dieharder [37] test suites, enabling rigorous validation before deployment in security-critical applications. Institutions can maintain audit trails documenting entropy provenance for compliance with security certifications.

Before enumerating the protections, it is useful to state the trust assumptions explicitly. QRNG-DD treats the QRNG appliance, the Collector, and the Gateway as trusted components operated by the institution within its administrative boundary; the network connecting them and the API clients consuming entropy from the Gateway are treated as untrusted. The security architecture is therefore designed to protect the trusted components and the entropy in transit between them against an adversary controlling the network or impersonating a client. Within this model, QRNG-DD implements five defense layers: physical security of the QRNG appliance in a secured facility, network isolation through firewall rules that block inbound connections to the Collector, TLS 1.3 encryption for all communications, four-layer packet integrity verification (HMAC-SHA256, CRC32, timestamps, sequence numbers) on the Collector-Gateway link, and application-level protection through API key authentication with rate limiting on the Gateway-Client link. The Collector presents no external attack surface as it has no listening sockets and initiates only outbound connections. The Gateway exposes authenticated REST endpoints protected by bearer tokens and configurable rate limits. This layered approach ensures that compromise of any single mechanism does not defeat the overall security posture. Detailed security analysis including threat model, attack surface enumeration, and compliance mapping is available in the supplementary materials.

QRNG-DD offers affordable quantum entropy infrastructure through MIT-licensed open-source software requiring no hardware beyond existing QRNG appliances. This matters for resource-constrained academic institutions where hardware data diode costs would consume entire equipment budgets. Benchmark testing confirms sub-4 ms median end-to-end latency on local networks, roughly 100 × faster than public services at 450 ms. Gateway processing runs under 100µs regardless of client location, with network round-trip time dominating end-to-end latency for remote clients. Support for megabyte-scale requests versus public service 1 KB limits enables high-volume research workloads without request fragmentation. The MCP integration supports AI-assisted research workflows absent from commercial appliances or public services, positioning QRNG-DD as infrastructure for the emerging paradigm of autonomous scientific agents (Table 1).

We validated statistical quality using Monte Carlo π estimation, which consistently meets NIST SP 800–22 criteria across millions of samples. Performance benchmarks were conducted on the following infrastructure: the Collector runs on a 4-core Intel KVM virtual machine with 16 GB RAM under Ubuntu 24.04 LTS, connected to an ID Quantique Quantis QRNG appliance on the university network; the Gateway runs on a 4-core AMD EPYC KVM virtual machine with 8 GB RAM under Ubuntu 24.04 LTS, hosted at a commercial provider (Contabo); local Docker benchmarks used an Intel Core i9–13,900 K workstation with 32 GB RAM under Windows 11 Pro. All Rust binaries were compiled in release mode with maximum optimization (opt-level 3, fat LTO, single codegen unit). Local network testing demonstrates 28.74 req/s sustained throughput with P50=3.62 ms, P95=6.89 ms, P99=9.13 ms latency and 100% success rate across extended 24-hour test runs. Burst capability reaches 438 req/s, constrained by QRNG appliance entropy generation (~80 KB/s) rather than software processing. Remote access verification at qrng.dataman.ro confirmed Gateway processing under 100µs, with end-to-end latency dominated by network round-trip time. These benchmarks demonstrate negligible overhead compared to direct appliance access while enabling cross-network distribution. The system has operated continuously for over six months without service interruption.

5. Conclusions

QRNG-DD delivers open-source research infrastructure for quantum entropy distribution, addressing needs in quantum computing, AI-assisted research, and computational physics. The software-based data diode architecture achieves sub-4 ms median latency with negligible processing overhead, while MCP integration allows AI agents to access quantum randomness through standardized tools. Multi-source entropy aggregation addresses vendor backdoor concerns through information-theoretic mixing guarantees.

We acknowledge several limitations. Software data diode architecture offers adequate isolation for research environments but weaker guarantees than hardware data diodes. The security depends on correct firewall configuration and operating system isolation rather than physical constraints. Organizations with critical infrastructure requirements should employ physical data diodes; QRNG-DD suits research and moderate-security deployments. Throughput is constrained by QRNG appliance entropy generation, typically ~80 KB/s for single Quantis units. Applications requiring higher throughput need multiple appliances with multi-source aggregation; the software layer adds negligible overhead. The Collector-Gateway link uses HMAC-SHA256 for cryptographic authentication of entropy packets, while the Gateway-Client link relies on standard HTTPS with bearer token authentication, consistent with how public QRNG services and REST APIs operate; end-to-end cryptographic provenance from QRNG source to final client using asymmetric signatures remains a future extension. The Model Context Protocol was released in late 2024; while adoption grows through Claude, ChatGPT, and open-source agents, the ecosystem remains young. We monitor protocol evolution and maintain compatibility as the specification matures.

We release the complete implementation under MIT license, offering transparency that allows researchers to validate entropy provenance and extend the system for domain-specific applications. Future development includes additional QRNG vendor integration (ComScire, QuintessenceLabs), federated entropy distribution for multi-institution collaborations enabling shared quantum randomness pools, and enhanced

Table 1
Comparison with Related Quantum Entropy Access Solutions.

Feature	Public QRNG (ANU)	NIST Beacon [13]	libqrng [24]	QRNG-DD	Existing randomness MCP servers
Network Access	Internet required	Internet required	Local only	Cross-boundary	Internet/local
Rate Limits	*5 req/sec*	1 pulse/min	Appliance limit	Appliance limit	N/A
Request Size	1024 bytes max	512 bits/pulse	Configurable	Megabyte scale	Small
Latency	~450ms	~60 s (pulse interval)	<1 ms (local)	<10 ms P99	Variable
Privacy	Centralized logs	Public record	Private	Private	Varies
Security Model	HTTPS	Signed pulse	Direct appliance	Data diode	None
AI Integration	None	None	None	MCP native	MCP native
Language	N/A	N/A	C	Rust	Various
Memory Safety	N/A	N/A	Manual	Compile-time	N/A
Multi-source	No	No	No	Yes (XOR/HKDF)	No

statistical testing integration with automated quality reports. The combination of software data diode security, AI agent integration, and high-performance Rust implementation addresses a gap in available research infrastructure for quantum randomness distribution.

Data availability statement

The QRNG-DD implementation, including all source code, example applications, and benchmark artifacts, is available under MIT license at https://github.com/vbocan/qrng-data-diode.

Funding

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Declaration of generative AI and AI-assisted technologies in the writing process

During the preparation of this work, the author used Claude (Anthropic) to improve the language and readability of the manuscript. After using these tools, the author reviewed and edited the content as needed and takes full responsibility for the content of the publication.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The author gratefully acknowledges Politehnica University of Timisoara for providing access to the Quantis appliance used in developing and testing this system.

Supplementary materials

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.softx.2026.102676.

References

[1] Herrero-Collantes M, Garcia-Escartin JC. Quantum random number generators. Rev Mod Phys 2017;89(1):015004. https://doi.org/10.1103/RevModPhys.89.015004. article.
[2] Stipčević M, Koç ÇK. True random number generators. Open problems in mathematics and computational science. Springer; 2014. p. 275–315. https://doi.org/10.1007/978-3-319-10683-0_12.
[3] Petrov M, Radchenko I, Steiger D, Renner R, Troyer M, Makarov V. Independent quality assessment of a commercial quantum random number generator. EPJ Quantum Technol 2022;9(1):17. https://doi.org/10.1140/epjqt/s40507-022-00136-z. article.
[4] Ma X, Yuan X, Cao Z, Qi B, Zhang Z. Quantum random number generation. npj Quantum Inf 2016;2:16021. https://doi.org/10.1038/npjqi.2016.21. article.
[5] Schneier B. Applied cryptography: protocols, algorithms, and source code in c. 2nd ed. New York: John Wiley & Sons; 1996. ISBN: 978-0471117094.
[6] Rubinstein RY, Kroese DP. Simulation and the monte carlo method. 3rd ed. Hoboken, NJ: John Wiley & Sons; 2016. https://doi.org/10.1002/9781118631980.
[7] Eastlake D, Schiller J, Crocker S. Randomness requirements for security. RFC 4086 Internet Eng Task Force 2005 [Online]. Available: https://datatracker.ietf.org/doc/html/rfc4086 [Accessed: 02-Jan-2026].
[8] I.D. Quantique, "Quantis QRNG appliance," 2024. [Online]. Available: https://www.idquantique.com/random-number-generation/products/quantis-rng-appliance/. [Accessed: 02-Jan-2026].
[9] Preskill J. Quantum computing in the NISQ era and beyond. Quantum 2018;2:79. https://doi.org/10.22331/q-2018-08-06-79.
[10] Owl Cyber Defense, "Data diode technology," 2024. [Online]. Available: https://owlcyberdefense.com/resource/data-diode-security-solutions/. [Accessed: 02-Jan-2026].
[11] Stouffer K, Pillitteri V, Lightman S, Abrams M, Hahn A. Guide to Industrial Control Systems (ICS) Security. NIST special publication 800-82 rev. 2. National Institute of Standards and Technology; 2015. https://doi.org/10.6028/NIST.SP.800-82r2.
[12] Australian National University, "Anu Qrng api," 2024. [Online]. Available: https://qrng.anu.edu.au/. [Accessed: 02-Jan-2026].
[13] National Institute of Standards and Technology, "NIST randomness beacon," 2024. [Online]. Available: https://beacon.nist.gov/. [Accessed: 02-Jan-2026].
[14] Wang L, Ma C, Feng X, Zhang Z, Yang H, Zhang J, et al. A survey on large language model based autonomous agents. Front Comput Sci 2024;18(6):186345. https://doi.org/10.1007/s11704-024-40231-1. article.
[15] Jung R, Jourdan JH, Krebbers R, Dreyer D. RustBelt: securing the foundations of the Rust programming language. In: Proceedings of the ACM on Programming Languages. 2; 2018, 66. https://doi.org/10.1145/3158154. article.
[16] Matsakis ND, Klock II FS. The Rust Language. ACM SIGAda Ada Lett 2014;34(3):103–4. https://doi.org/10.1145/2692956.2663188.
[17] Tokio Contributors, "Tokio: an asynchronous runtime for the Rust programming language," 2024. [Online]. Available: https://tokio.rs/. [Accessed: 02-Jan-2026].
[18] Anthropic, "Model context protocol specification," 2024. [Online]. Available: https://modelcontextprotocol.io/. [Accessed: 02-Jan-2026].
[19] Shannon CE. Communication Theory of secrecy systems. Bell Syst Tech J 1949;28(4):656–715. https://doi.org/10.1002/j.1538-7305.1949.tb00928.x.
[20] Nielsen MA, Chuang IL. Quantum computation and quantum information, 10th anniversary editor. Cambridge: Cambridge University Press; 2010. https://doi.org/10.1017/CBO9780511976667.
[21] Glorot X, Bengio Y. Understanding the difficulty of training deep feedforward neural networks. In: Proc. 13th Int. Conf. Artificial Intelligence and Statistics (AISTATS); 2010. p. 249–56.
[22] Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res 2014;15(1):1929–58.
[23] Metropolis N, Ulam S. The Monte Carlo method. J Am Stat Assoc 1949;44(247):335–41. https://doi.org/10.1080/01621459.1949.10483310.
[24] Ardelean SM, Udrescu M, Stangaciu V. Easy to integrate API for accessing true random numbers generated with IDQ's Quantis Appliance. SoftwareX 2024;27:101841. https://doi.org/10.1016/j.softx.2024.101841. article.
[25] Klabnik S, Nichols C. The rust programming language. 2nd ed. San Francisco: No Starch Press; 2023.
[26] Ginter A. Unidirectional security gateways: stronger than firewalls. In: Proc. 14th Int. Conf. Accelerator and Large Experimental Physics Control Systems (ICALEPCS'13); 2013. paper THCOBA02. [Online]. Available: https://proceedings.jacow.org/ICALEPCS2013/papers/thcoba02.pdf [Accessed: 02-Jan-2026].
[27] Krawczyk H, Bellare M, Canetti R. HMAC: keyed-hashing for message authentication. RFC 2104 Internet Eng Task Force 1997 [Online]. Available: https://datatracker.ietf.org/doc/html/rfc2104 [Accessed: 02-Jan-2026].
[28] Fielding RT. Architectural styles and the design of network-based software architectures. Irvine: University of California; 2000. Ph.D. dissertation.
[29] Krawczyk H, Eronen P. HMAC-based extract-and-expand key derivation function (HKDF). RFC 5869 Internet Eng Task Force 2010 [Online]. Available: https://datatracker.ietf.org/doc/html/rfc5869 [Accessed: 02-Jan-2026].
[30] Bird JJ, Ekárt A, Faria DR. On the effects of pseudorandom and quantum-random number generators in soft computing. Soft Comput 2020;24:9243–56. https://doi.org/10.1007/s00500-019-04450-0.
[31] Boiko DA, MacKnight R, Kline B, et al. Autonomous chemical research with large language models. Nature 2023;624:570–8. https://doi.org/10.1038/s41586-023-06792-0.
[32] Miller M. Trends, challenges, and strategic shifts in the software vulnerability mitigation landscape. Microsoft Security Response Center; 2019 [Online]. Available: https://github.com/microsoft/MSRC-Security-Research/blob/master/presentations/2019_02_BlueHatIL/ [Accessed: 02-Jan-2026].
[33] "Memory safety, " the chromium projects, 2024. [Online]. Available: https://www.chromium.org/Home/chromium-security/memory-safety/[Accessed: 02-Jan-2026].
[34] Ferrenberg AM, Landau DP, Wong YJ. Monte Carlo simulations: hidden errors from 'Good' Random number generators. Phys Rev Lett 1992;69(23):3382–4. https://doi.org/10.1103/PhysRevLett.69.3382.
[35] Hastings WK. Monte Carlo sampling methods using Markov chains and their applications. Biometrika 1970;57(1):97–109. https://doi.org/10.1093/biomet/57.1.97.
[36] Rukhin A, et al. A statistical test suite for random and pseudorandom number generators for cryptographic applications. NIST special publication 800-22 rev. 1a. National Institute of Standards and Technology; 2010 [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf [Accessed: 02-Jan-2026].
[37] R.G. Brown, "Dieharder: a random number test suite," 2024. [Online]. Available: https://webhome.phy.duke.edu/~rgb/General/dieharder.php. [Accessed: 02-Jan-2026].