BRIEF DESCRIPTION OF THE DRAWINGS

Docket Number: RUTHERFORD-012-PROV
Total Sheets: 7

FIGURE 1 - System Architecture Overview

Sheet 1 of 7

Figure 1 illustrates the overall system architecture of the defensive cybersecurity testing framework showing the relationship between the vulnerability discovery system and existing PQC implementations. The diagram shows:

- Input layer (100) accepting implementations from cuPQC (102), LibOQS (104), and other PQC libraries (106)
- GPU-accelerated testing core (110) with CUDA kernels (112) and tensor cores (114)
- Vulnerability discovery engine (120) with attack simulation (122) and pattern recognition (124)
- Compliance validation layer (130) supporting multiple standards
- Output layer (140) generating vulnerability reports (142), compliance certificates (144), and migration recommendations (146)

FIGURE 2 - CUDA Kernel Architecture

Sheet 2 of 7

Figure 2 depicts the CUDA kernel architecture optimized for parallel quantum attack simulation:

- Thread block organization (200) with 1024 threads per block
- Shared memory allocation (210) for attack state storage
- Warp-level primitives (220) for synchronized vulnerability detection
- Global memory access patterns (230) optimized for coalesced reads
- Early termination logic (240) triggered upon vulnerability discovery
- Attack vector distribution (250) across streaming multiprocessors

---

FIGURE 3 - Tensor Core Optimization

Sheet 3 of 7

Figure 3 shows tensor core optimization for adversarial syndrome decoding attacks:

- Matrix multiplication units (300) configured for cryptanalysis
- Tensor core array (310) with 640 tensor cores per SM
- Mixed precision operations (320) using FP16/INT8
- Syndrome matrix decomposition (330) for lattice attacks
- Parallel correlation computation (340) for side-channel analysis
- Performance metrics display (350) showing 10-100x speedup

FIGURE 4 - Vulnerability Discovery Data Flow

Sheet 4 of 7

Figure 4 illustrates the vulnerability discovery data flow and automated weakness identification:

- Implementation ingestion module (400)
- Static analysis pipeline (410) for code inspection
- Dynamic testing framework (420) with runtime monitoring
- Side-channel measurement apparatus (430)
- Quantum attack simulator (440) implementing Grover's and Shor's algorithms
- Vulnerability classification system (450) categorizing findings by severity
- Automated reporting engine (460)

FIGURE 5 - Multi-Standard Compliance Pipeline

Sheet 5 of 7

Figure 5 presents the multi-standard compliance report generation pipeline:

- Standard parsers (500) for NIST FIPS 203/204/205
- ETSI TR 103 619 validator (510)
- ISO/IEC 18033-2 checker (520)
- Common Criteria EAL4+ evaluator (530)
- Unified compliance engine (540) cross-referencing requirements
- Report generator (550) with customizable templates
- Certification issuance module (560)

FIGURE 6 - Migration Recommendation Algorithm

Sheet 6 of 7

Figure 6 depicts the migration recommendation algorithm based on Mosca's theorem:

- Input parameters (600): X (data sensitivity), Y (migration time), Z (quantum threat)
- Risk calculation engine (610) implementing X + Y >= Z
- Vulnerability assessment integration (620)
- Timeline generator (630) with milestone tracking
- Priority matrix (640) ranking systems by risk
- Recommendation engine (650) with actionable steps
- Dashboard visualization (660)

FIGURE 7 - Implementation Layer Separation

Sheet 7 of 7

Figure 7 shows the architectural separation between implementation libraries and testing layer:

- Implementation layer (700) containing cuPQC (702), LibOQS (704), DPCrypto (706)
- API abstraction layer (710) providing unified interface
- Testing framework (720) operating independently
- Vulnerability discovery modules (730) targeting each implementation
- Results aggregation layer (740)
- Feedback loop (750) to implementation providers
- Clear boundary (760) between defensive testing and cryptographic operations

---

DRAWING CONVENTIONS

All figures use the following conventions:

- Solid lines indicate data flow
- Dashed lines indicate control flow
- Dotted lines indicate optional connections
- Shaded boxes represent GPU-accelerated components
- Double borders indicate security-critical modules

- Reference numerals are consistent across all figures

Total Pages: 2

End of Drawing Descriptions