**QSC - High-Security Cryptographic Library for Critical Domains**

A brief summary of the security architecture and standards compliance with industry-recognized coding and module-validation standards.

Revision 1a, May 23, 2025

John G. Underhill – [contact@qrcscorp.ca](mailto:contact@qrcscorp.ca)

**Abstract**  
This paper presents the design goals, compliance requirements, and defensive-security focus of the QSC (Quantum Secure Cryptographic) library project. QSC targets applications in medical, military, financial, automotive, and avionics systems; industries where failure is not an option and undefined behavior is outlawed. We describe the dual‐build approach (QSC and QCM), enumerate the standards and coding rules to which QSC must conform, and explain how we balance strict compliance with real-world performance needs.

**1. Introduction**

Security-critical systems demand cryptographic components that are not only functionally correct, but also immune to subtle implementation flaws, timing channels, and undefined-behavior vulnerabilities. QSC is a portable, standards-driven C library providing post-quantum and classical primitives designed for deployment in environments governed by the most stringent safety and security regulations.

Our objectives are:

* **Portability:** One codebase, four major platforms (macOS, Linux, BSD, Windows).
* **Defensive Security:** Eliminate undefined and implementation-dependent behavior; enforce constant-time operations; detect and prevent logic errors.
* **Standards Compliance:** Align with industry-recognized coding and module-validation standards.
* **Real-World Performance:** Support high-throughput use cases (e.g. GB-scale buffer operations, SIMD acceleration) without sacrificing compliance.

**2. Compliance and Portability Requirements**

QSC’s quality-assurance regimen includes the following mandatory checks. Each rule set is linked to its authoritative home page:

* **Portability (POSIX):**  
  Adhere to the Portable Operating System Interface (POSIX) specification to ensure consistent behavior across UNIX-like systems and Windows subsystems.  
  <https://pubs.opengroup.org/onlinepubs/9699919799/>
* **MISRA C:2023 Compliance**  
  Enforce guidelines to eliminate undefined and critical-unspecified behavior in safety-critical C code.  
  <https://www.misra.org.uk/>
* **CERT C Secure Coding Standard**  
  Detect and prevent logic errors, buffer overflows, and misuse of the C language that lead to vulnerabilities.  
  <https://wiki.sei.cmu.edu/confluence/display/c/SEI+CERT+C+Coding+Standard>
* **FIPS 140-3 Cryptographic Module Validation**  
  Validate module design, operational environment, and self-tests under FIPS 140-3 for U.S. government and regulated industries.  
  [Cryptographic Module Validation Program | CSRC](https://csrc.nist.gov/Projects/Cryptographic-Module-Validation-Program/Validated-Modules)
* **Common Criteria (ISO/IEC 15408)**  
  Achieve assurance levels (EAL) through standardized evaluation of security functionality and design.  
  <https://www.commoncriteriaportal.org/>
* **ISO 26262 Functional Safety**  
  Satisfy automotive safety integrity levels (ASIL) for software within electrical/electronic systems.  
  <https://www.iso.org/standard/68383.html>
* **DO-178C Software Considerations in Avionics**  
  Comply with airborne software assurance levels (DAL) for certification in avionics systems.  
  [Your Complete DO-178C Guide to Aerospace Software Compliance - LDRA](https://ldra.com/do-178/)
* **Common Criteria EAL Levels**  
  Specify the required Evaluation Assurance Level (EAL) per deployment; from EAL1 (functionally tested) up to EAL7 (formally verified design).  
  <https://www.commoncriteriaportal.org/>
* **Logic-Error and Coding-Mistake Detection**  
  Apply static analysis, peer code review, and unit-testing to uncover logic flaws and implementation mistakes early in the development cycle.  
  <https://wiki.sei.cmu.edu/confluence/display/c/SEI+CERT+C+Coding+Standard>
* **Constant-Time Operation Enforcement**  
  Ensure that all cryptographic primitives execute in time independent of secret-data values, preventing timing side-channels.  
  [Guidelines for Mitigating Timing Side Channels Against Cryptographic...](https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/secure-coding/mitigate-timing-side-channel-crypto-implementation.html)

**3. Dual-Build Strategy: QSC and QCM**

* **QSC (Full-Feature Library):**
  + Includes platform APIs for file I/O, console I/O, OS threading, dynamic memory, and SIMD intrinsics (where available).
  + Enables high-performance servers and applications that allocate large buffers (e.g. several GB) without page-fault storms.
  + Nearly MISRA-compliant: impeachable features (dynamic allocation, OS calls, intrinsics) are isolated in well-documented modules.
* **QCM (Quantum Cryptographic Modules):**
  + A strictly MISRA-certifiable static library containing only primitives drawn from NIST and ISO standards:
    - **AES:** GCM, CBC, CTR, ECB
    - **SHA-2:** HMAC, HKDF
    - **SHA-3 Family:** SHAKE, cSHAKE, KMAC
    - **ECC:** ECDSA, ECDH
    - **Post-Quantum Primitives:** Dilithium, Kyber, SPHINCS+, Falcon, McEliece
    - **ISO-Standardized:** ChaCha, Poly1305
  + No reliance on dynamic memory, file/console I/O, or platform-specific intrinsics.

QCM can undergo full MISRA certification and satisfy FIPS 140-3, Common Criteria, ISO 26262, and DO-178C assessments without waivers.

**4. Defensive-Security Emphasis**

Cryptographic libraries used in web servers and general-purpose applications prioritize throughput and feature breadth, often tolerating:

* Undefined or implementation-defined behavior.
* Occasional use of dynamic memory or OS APIs without rigorous bounds checking.
* Potential timing and cache side-channels under adversarial conditions.

By contrast, QSC enforces:

1. **Zero Undefined Behavior:** Every C construct is vetted against the standard; file and OS API calls are encapsulated and validated before use.
2. **Rigorous Static Analysis:** Automated tools for MISRA, CERT C, and bespoke logic checks run on every commit.
3. **Constant-Time Guarantees:** Secrets never influence control flow or memory access patterns.
4. **Comprehensive Self-Tests:** Each module includes power-up and runtime tests to detect tampering, hardware faults, or configuration errors.
5. **Assessed Integration Points:** OS-specific code is confined to thin, peer-reviewed layers, facilitating system-level certification without re-auditing core cryptographic routines.

**5. Design Considerations and Performance**

* **SIMD Intrinsics:** Enabled in QSC for entropy-expensive or high-throughput routines (e.g. hashing large datasets), with fallbacks to portable C.
* **Dynamic Memory:** Permitted only in QSC’s performance critical sections; QCM bypasses all heap usage.
* **Stack Allocation Limits:** Avoid multi-MB stack frames to prevent page thrashing; large buffers are heap-backed or mapped via OS APIs under strict bounds.
* **Threading and Concurrency:** Use only POSIX or Windows threading APIs; all shared mutable state is protected by MISRA-approved synchronization primitives.

These choices allow QSC to deliver industrial-strength performance while maintaining a verifiable compliance pedigree.

**6. Conclusion**

The QSC project delivers a new generation of cryptographic software tailored for the most demanding safety and security-critical domains. By adopting a dual-build strategy (QSC and QCM), we reconcile the need for high performance with the imperative of uncompromising standards compliance, meeting MISRA C:2023, CERT C, FIPS 140-3, Common Criteria/EAL, ISO 26262, and DO-178C. Defensive engineering practices, constant-time guarantees, and exhaustive static analysis ensure that undefined behavior and implementation flaws are eradicated, providing a foundation upon which medical devices, military systems, financial services, and avionics can build their security architectures with confidence.