Architecting STAC

How ExeQuantum Technically Delivers Sovereignty, Transparency, Agility & Compliance

Samuel Tseitkin, CEO
Raymond K. Zhao, CTO

Contents

1 Executive Summary

The global transition to post-quantum cryptography (PQC) demands implementations that are not only fast but also provably resistant to side-channel attacks and compiler-induced vulnerabilities. Most high-assurance and / or verifiable implementations today fall into two categories:

- **C-heavy implementations** (e.g., Microsoft's HACL* [ZBPB17] and EverCrypt [PPF+20]): Developer-friendly, providing human-readable code, but reliant on compilers that may introduce subtle side-channel leaks.
- **Assembly-heavy implementations** (e.g., Formosa Crypto[1] written in Jasmin [ABB+17]): formally verifiable at the instruction level, but difficult to maintain, slower for non-critical routines, and less practical to integrate.

ExeQuantum introduces a **hybrid approach**. All subroutines involving secret values are written in Jasmin, ensuring constant-time execution and side-channel resistance at the instruction level by construction. Public-only operations remain in C, where compiler optimizations improve performance. This creates a clear boundary: Jasmin for security-critical paths and C for non-sensitive routines.

This architecture delivers three major advances:

1. **Stronger assurance** than C-heavy approaches: Jasmin eliminates compiler-related side-channel risks.
2. **Performance parity or better**: in some cases, our Jasmin implementations outperform C-based code (e.g., ML-DSA signing).
3. **Clearer auditability**: security evaluators can assume that every Jasmin value is secret, which accelerates certification.

[1] https://github.com/formosa-crypto

Guided by the STAC principles of ExeQuantum (Sovereignty, Transparency, Agility, Compliance), this hybrid methodology sets a new benchmark.

2 Background and Motivation

The NIST Post-Quantum Cryptography (PQC) standardization effort has defined robust algorithms to secure communications against the threat of quantum computers. However, the practical security of PQC depends not only on the mathematical soundness of the algorithms, but also on the correctness and resilience of their implementations.

Side-channel attacks are a particular concern. Even if the algorithm itself is cryptographically sound, timing differences or microarchitectural leakage in the implementation can reveal secret keys. For this reason, high-assurance PQC implementations must guarantee *side-channel resistant execution* for all operations involving secret values.

Today, there are two dominant approaches:

2.1 C-heavy Implementations

Most industrial-grade implementations, including HACL* and EverCrypt by Microsoft Research, provide high-assurance and / or verifiable code at the C level. This provides human-readable code, good developer ergonomics, high portability, and allows compilers to optimize for performance. However, mainstream compilers such as MSVC, GCC, and Clang are not designed to preserve side-channel-resistant semantics. Even simple integer multiplications can be compiled into instructions whose execution time varies depending on input [KPVV16]. This creates the risk of side-channel leakage, even if the source code appears to be side-channel resistant. This behavior of the compiler also varies with different compiler versions and optimization levels[1]. Recompilation after compiler upgrade or

[1] https://github.com/veorq/cryptocoding

change of compiler flags may generate vulnerable assembly code, even after the previously generated assembly code is checked for side-channel vulnerabilities.

Some libraries in this category, e.g. EverCrypt, also use assembly. However, their assembly code is usually employed at a very limited scope, and the purpose may not be side-channel resistant. For example, EverCrypt only uses assembly code in performance-critical operations by leveraging architecture-specific instructions (e.g. AES-NI) for speed rather than for assurance[2].

2.2 Assembly-heavy Implementations

On the other end of the spectrum, projects such as Everest and other open-source Jasmin repositories such as Formosa Crypto implement nearly all routines directly in assembly or domain-specific assembly languages. This eliminates the risk of compiler-induced side-channel leaks and allows formal verification at the instruction level. However, these implementations are difficult to maintain at scale, slower in non-critical routines where compiler optimizations would be beneficial, and less practical for integration into existing systems written in other (high-level) programming languages.

2.3 The Gap

This creates a gap in the PQC ecosystem. C-heavy approaches are fast and convenient but weaker in assurance. Assembly-heavy approaches are secure and verifiable at the instruction level but are harder to integrate and maintain.

The challenge is to develop an approach that achieves the best of both worlds:

- Eliminate compiler risks in all *security-critical* subroutines.
- Retain C where it is *safe* to do so, in order to maximize performance and maintainability.
- Provide a clear and auditable boundary between critical and non-critical code.

ExeQuantum's work directly addresses this challenge.

[2] https://hacl-star.github.io/HaclValeEverCrypt.html; https://hacl-star.github.io/Supported.html

3 ExeQuantum's Hybrid Approach

ExeQuantum introduces a principled hybrid design that combines the strengths of Jasmin and C while avoiding the weaknesses of either approach. The guiding principle is simple:

> *Every operation involving secret data must be implemented in Jasmin; every operation involving only public data may remain in C.*

3.1 Jasmin Layer: Security-Critical Code

Jasmin is a low-level, domain-specific language designed for cryptographic implementations. It enforces constant-time execution and enables formal verification at the assembly level. By writing *all subroutines that touch secret values* in Jasmin, ExeQuantum ensures:

- Constant-time execution by construction, eliminating timing side-channel vulnerabilities.
- Formal proofs of safety and correctness at the instruction level.
- Side-channel resistance that does not depend on the compiler behavior.

3.2 C Layer: Public-Value Code

For subroutines that operate exclusively on public values, ExeQuantum uses C. This allows modern compilers to perform aggressive optimizations without compromising security. Benefits include:

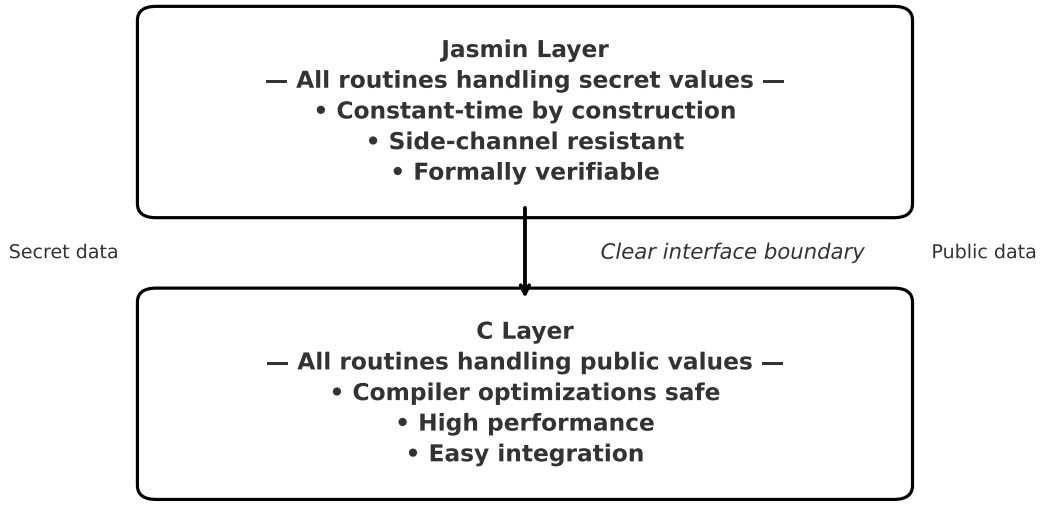

Figure 3.1: ExeQuantum's hybrid architecture: Jasmin for all secret-handling routines; C for public-only routines, with a clear interface boundary.

- Higher performance for non-critical routines.
- Easier integration with APIs, SDKs, and existing systems.
- Improved maintainability for developers who interact with the codebase.

3.3 Clear Security Boundary

This design creates a **natural and auditable boundary** between critical and non-critical code:

- Every value inside Jasmin can be assumed to be secret, simplifying side-channel evaluation.
- Every value in the C layer is public, allowing compilers to optimize freely.

- Auditors and certification bodies can independently evaluate each layer with clear trust assumptions.

3.4 Integration by Design

To maximize usability, ExeQuantum preserves standard C interfaces at the boundary between Jasmin and C. This allows our implementations to be easily incorporated into higher-level APIs, cloud services, or SDKs without requiring specialized tooling.

3.5 Benefits

In practice, this hybrid approach achieves the following:

1. **Assurance:** side-channel safe by construction for all secret handling codes.
2. **Performance:** compiler-optimized C for public routines ensures efficiency.
3. **Maintainability:** clearer division of responsibilities for developers and auditors.
4. **Agility:** simple integration with existing C-based cryptographic ecosystems.

This hybrid architecture, guided by the STAC principles (Sovereignty, Transparency, Agility, Compliance), represents a new model for high-assurance PQC implementations.

4 Technical Achievements

ExeQuantum's implementation covers the NIST-standardized algorithms **ML-KEM** and **ML-DSA**, delivering both stronger assurance and competitive performance. The following milestones highlight the distinctive advantages of our approach.

4.1 Extended Verified Coverage

Unlike C-heavy implementations (e.g., HACL* and EverCrypt), ExeQuantum writes *every subroutine that handles secret values* directly in Jasmin. This ensures constant-time execution across the full critical path, not only in a subset of routines. The result is a more comprehensive and systematic elimination of side-channel risks.

4.2 Last-Mile Protection

By controlling the final translation step to assembly, our approach closes what we term the *"last mile"* gap between high-level C code and low-level instructions. Traditional C implementations must trust compilers not to introduce side-channel vulnerability, but in practice compilers are not designed to provide such guarantees. With Jasmin, the final assembly is formally validated, removing this uncertainty.

4.3 Performance Gains

A key finding of our development is that assurance does not imply performance trade-offs. In fact, for some operations, Jasmin implementations outperform C equivalents. For example:

- **ML-DSA signing:** Our Jasmin implementation achieves higher throughput than the C-based reference implementation.
- **ML-KEM key generation and decapsulation:** We observe performance on par with the reference C code, while retaining formal verification guarantees.

Table 4.1: ML-DSA performance: Original vs. Hardened implementations

Operation	Original	Hardened	Throughput
ML-DSA-44			
KeyGen	17094	19740	115.48%
Sign	5085	6303	123.95%
Verify	16303	19581	120.11%
ML-DSA-65			
KeyGen	9341	11500	123.11%
Sign	2881	3698	128.37%
Verify	9969	11808	118.44%
ML-DSA-87			
KeyGen	6096	7087	116.27%
Sign	2146	2945	137.23%
Verify	5822	7148	122.77%

4.4 Safety Checks

All Jasmin routines undergo systematic safety validation:

- **Side-channel resistance:** verified constant time semantics across the entire secret-handling path.
- **Memory safety:** explicit checks to prevent out-of-bounds access and other low-level vulnerabilities.
- **Determinism:** reproducibility guarantees for cryptographic outputs under identical inputs.

4.5 Integration and Maintainability

Despite the high level of assurance, our implementation preserves the ergonomics of the developer. By exposing standard C interfaces at the boundary, we ensure:

- Drop-in integration with APIs and SDKs.
- Compatibility with cloud-native services and performance benchmarking tools.
- Easier onboarding for developers accustomed to C-based cryptographic libraries.

4.6 Summary

Through extended Jasmin coverage, last-mile protection, verified safety checks, and demonstrated performance parity (and in some cases superiority), ExeQuantum has achieved a milestone: **a PQC implementation that is simultaneously verifiable, high-performance, and practical to deploy**.

5 Implications for High-Assurance Environments

The value of ExeQuantum's hybrid architecture is most evident in domains where both security and compliance are nonnegotiable. By establishing a strict boundary between Jasmin and C, our design provides clarity to auditors and regulators while providing practical performance to operators.

5.1 Healthcare

Healthcare systems manage highly sensitive patient data that is both a privacy target and a regulatory concern.

- The Jasmin boundary ensures that all cryptographic operations that touch confidential health information are constant-time and side-channel safe.
- This makes certification under health data regulations (such as HIPAA or GDPR health provisions) more straightforward because evaluators can clearly scope their analysis.
- Agility is maintained: C-layer routines allow for efficient integration into electronic health record systems, telemedicine platforms, and secure medical IoT devices.

5.2 Finance

Financial institutions are required to adopt cryptographic standards that are verifiable, maintainable, and often independently auditable.

- Our approach provides **transparency**: the Jasmin code path is formally verifiable and side-channel resistant.
- It supports **compliance**: regulators can treat Jasmin routines as the complete scope of high-assurance review, reducing certification complexity.
- Performance parity with existing C code ensures that adoption does not disrupt latency-sensitive applications such as trading systems or payment networks.

5.3 Government and Sovereignty

For governments, the adoption of PQC is inseparable from sovereignty and trust.

- By eliminating compiler dependence in security-critical paths, states can independently verify the correctness of Jasmin routines without reliance on opaque toolchains.
- The clear Jasmin–C separation offers **sovereignty**: assurance that cryptographic protections can be independently audited and maintained.
- Transparency and agility are preserved, allowing integration into various government systems ranging from secure communications to national infrastructure.

5.4 Alignment with STAC

ExeQuantum's hybrid approach naturally reflects the STAC principles:

- **Sovereignty:** Jasmin routines can be independently verified without relying on foreign compilers or toolchains.
- **Transparency:** the boundary makes side-channel assurance auditable and certifiable.
- **Agility:** C interfaces preserve ease of integration into real-world systems.

- **Compliance:** certification bodies can scope evaluations cleanly, reducing cost and complexity.

In short, ExeQuantum delivers not only stronger technical assurance, but also a deployment model that simplifies regulatory adoption and accelerates PQC readiness in the most demanding environments.

6 Conclusion and Future Work

ExeQuantum's hybrid architecture demonstrates that high assurance and high performance are not competing goals. By writing every secret-handling routine in Jasmin while retaining C for public-only operations, we eliminate compiler-induced side channels, achieve extended verified coverage, and in some cases even surpass C-based implementations in speed.

This achievement represents a significant milestone in the global transition to post-quantum cryptography. For the first time, implementers can adopt PQC that is,

- **Verifiable:** constant-time and side-channel safe by construction.
- **Performant:** competitive with, and in some operations faster than C implementations.
- **Practical:** integrated through standard C interfaces, reducing friction for deployment.

6.1 Future Work

Our roadmap extends these advances beyond ML-KEM and ML-DSA:

- **Broader algorithm coverage:** expanding Jasmin coverage to additional NIST PQC algorithms, including signature and key encapsulation mechanisms such as FN-DSA, HQC and SLH-DSA.
- **Advanced assurance:** exploring device-bound signatures, constant-time randomness generation, and formally verified memory safety.
- **Applied innovation:** integrating PQC into verifiable AI systems, secure APIs, and cloud-native environments.

6.2 Invitation to Collaborate

We invite current clients, governments, healthcare providers, and financial institutions to partner with ExeQuantum. Pilot deployments will not only validate the assurance and performance of our implementations in production environments, but also shape the standards of what high-assurance PQC looks like in practice.

In alignment with our STAC framework (Sovereignty, Transparency, Agility, Compliance), ExeQuantum is committed to ensuring that the transition to post-quantum cryptography is both technically sound and practically achievable.

Bibliography

[ABB+17] José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, Arthur Blot, Benjamin Grégoire, Vincent Laporte, Tiago Oliveira, Hugo Pacheco, Benedikt Schmidt, and Pierre-Yves Strub. Jasmin: High-assurance and high-speed cryptography. In *CCS*, pages 1807–1823. ACM, 2017.

[KPVV16] Thierry Kaufmann, Hervé Pelletier, Serge Vaudenay, and Karine Villegas. When constant-time source yields variable-time binary: Exploiting Curve25519-donna built with MSVC 2015. In *CANS*, volume 10052 of *Lecture Notes in Computer Science*, pages 573–582, 2016.

[PPF+20] Jonathan Protzenko, Bryan Parno, Aymeric Fromherz, Chris Hawblitzel, Marina Polubelova, Karthikeyan Bhargavan, Benjamin Beurdouche, Joonwon Choi, Antoine Delignat-Lavaud, Cédric Fournet, Natalia Kulatova, Tahina Ramananandro, Aseem Rastogi, Nikhil Swamy, Christoph M. Wintersteiger, and Santiago Zanella-Béguelin. EverCrypt: A fast, verified, cross-platform cryptographic provider. In *SP*, pages 983–1002. IEEE, 2020.

[ZBPB17] Jean Karim Zinzindohoué, Karthikeyan Bhargavan, Jonathan Protzenko, and Benjamin Beurdouche. HACL*: A verified modern cryptographic library. In *CCS*, pages 1789–1806. ACM, 2017.