

A Cryptographic Agility Framework for Post-Quantum Secure API Migration: Empirical Performance Analysis and Architectural Blueprint

Gabriel Silva Atencio¹✉

¹Latin American University of Science and Technology (ULACIT), San José, Costa Rica.

Abstract

The public-key infrastructure is at risk because quantum computers will soon be useful for encryption. Still, there are a lot of things we don't fully understand about how Post-quantum cryptography (PQC) affects API ecosystems as a whole. This study is the first to do a thorough, organized evaluation of NIST-standard PQC algorithms used in a real-world microservices framework with full API security protocols (TLS 1.3, JWT, and OAuth 2.0). Previous studies only looked at isolated primitives or evaluations that only used TLS. A controlled testing platform shows a big difference in performance: PQC key encoding (Ky-ber768) only causes an 8.2% rise in latency, but Post-quantum digital signatures (Dilithium3) make the TLS handshake take 193.6% longer and slow down API traffic by 29.2%. FALCON-512, an alternative lattice-based signature scheme, reduces handshake latency by 18.3% compared to Dilithium3 (108.4 ms vs. 132.7 ms), demonstrating that algorithm selection significantly impacts system-level performance even at equivalent NIST security levels. To deal with these problems, a cryptographic agility framework is suggested and shown to cut migration work by 87.6%, or from 14.5 person-hours to 1.8, which is a real benefit that wasn't seen in previous theoretical approaches. It is also tried out a combined PQC-classical shift approach. It works, but it costs more to run (185% for key management and 120% for log processing) and has a 266.8% handshake overhead. These findings are synthesized into six actionable design principles for Post-quantum API security. This work transforms the PQC transition from a theoretical problem into an evidence-based systems engineering roadmap, enabling organizations to make quantified risk-performance trade-offs when migrating API infrastructure.

Keywords: application programming interface (API) security; cryptographic agility; microservices; post-Quantum cryptography; quantum computing; public-key infrastructure

Introduction

The security of the global digital environment of the 21st century is based on public-key cryptography. The computational intractability of problems such as the discrete logarithm problem and integer factorization ensures the security of online finance, e-commerce, secure messaging, and the Internet of Things (IoT) for classical computers. For decades, cryptographic systems such as Elliptic Curve Cryptography (ECC) and Rivest–Shamir–Adleman (RSA) have provided reliable security and are the foundation of the Public Key Infrastructure (PKI) of today. Nevertheless, the anticipated advent of cryptographically relevant quantum computers (CRQCs) poses a significant threat to the fundamental stability of this established paradigm.

The integer factorization problem and the discrete logarithm problem could be solved in polynomial time by a sufficiently powerful quantum computer, as demonstrated by Peter Shor’s 1994 quantum algorithm [1,2]. Consequently, the development of such a machine would render RSA, ECC, and the Digital Signature Algorithm (DSA) insecure. Although the precise timeline for achieving a cryptographically relevant quantum computer (CRQC) remains uncertain, continued advances in quantum hardware, together with expert assessments, national security roadmaps, and sustained research efforts, indicate the need for timely preparation [3]. For example, NIST’s Internal Report (IR) 8545 outlines a migration timeline extending to 2035, while the NSA’s Commercial National Security Algorithm Suite (CNSA) 2.0 mandates a post-quantum cryptography (PQC) transition for national security systems by 2033 [4]. This urgency is further reinforced by the “harvest now, decrypt later” attack model, in which adversaries capture and store encrypted data today with the intention of decrypting it once a CRQC becomes available [5].

Furthermore, the agreement is that cryptographic migration, especially for long-lived infrastructure, should begin soon to avoid systemic danger. This urgency is increased by the “harvest now, decrypt later” (HNDL) attack architecture, in which attackers capture and store encrypted data today with the purpose of decrypting it once a CRQC is available [3]. The quantum danger goes from being something that might happen in the future to being a real strategic risk right now. This is especially true for data that needs to be kept secret for a long time, like intellectual property, government secrets, and personal health information.

An important analytical finding is that the HNDL model has a big impact on the risk calculation for API ecosystems. Application Programming Interfaces (APIs) handle personal information, session keys, and access tokens all the time, and these things may still be valuable years or decades from now. Because of this, the idea that

“migration can wait until quantum computers arrive” is not based on facts. HNDL attacks can happen now, and more data can be lost for every day that migration is put off. Based on the HNDL danger model, this analysis approach makes the present study even more important. It’s different from earlier work that only looked at quantum migration in terms of the future.

Still, the switch to PQC will be very hard on technology, especially for systems that use spread software. In the quantum era, APIs are a critical and expansive attack surface. APIs facilitate data exchange, interoperability, and business functions across service-oriented designs, cloud-native architectures, and microservices [6]. These APIs are susceptible to quantum attacks, as they depend on frameworks such as JSON Web Tokens (JWT) and OAuth 2.0 for authentication and authorization, as well as on Transport Layer Security (TLS) for in-transit security. As a result, the security of APIs is an essential element of any practical PQC transition plan and is not an optional step.

PQC algorithms introduce more computationally intensive operations and larger key and signature sizes. In API systems that are subject to strict Service-Level Agreements (SLAs) and handle millions of requests per second, these attributes are directly translated into measurable impacts on scalability, infrastructure cost, throughput, and latency. Cryptographic agility, which is the capacity to transition between cryptographic algorithms and parameters with minimal system disruption, is crucial for managing this transition [7, 8]. Not only can an agile system accommodate the planned PQC migration, but it can also accommodate future cryptographic weaknesses or deprecations.

The theoretical gap in the current literature is structural rather than descriptive. Previous research has looked at how well PQC starters work on their own [9-12], but it hasn’t thought about how these extra costs move between the different levels of a production API stack. This is a big mistake because API speed isn’t just based on cryptographic processes; it is influenced by the interaction of TLS termination, token validation, gateway routing, backend processing, and database persistence. A signature verification that contributes 3 ms at the primitive level might contribute 30 ms at the system level when queuing effects, CPU saturation, and serialization delay are taken into account. By using a system-level analytical perspective, which views the API stack as a coherent socio-technical system rather than a collection of disparate parts, the present study fills this gap.

A thorough review of the research finds important gaps in the knowledge. A lot of research has been done on how well PQC methods work in theoretical security [13, 14] and protocol handshakes [9, 10]. However, not as much has been done on how to use PQC at the system level in real API situations. Most of the study being done right now is focused on single stack components or specific

performance measures. At the moment, there isn't a complete framework that connects cryptographic theory to software engineering and DevOps processes used in API systems in the real world. Also, the ideas behind cryptographic agility have been made very clear, but there aren't many real, tested building plans out there. Lastly, hybrid cryptography, which combines classical and PQC algorithms, has been talked about a lot at the protocol design level, but not much has been done to look at how it works and how hard it is to use in full API systems [15].

The lack of system-level analysis isn't just a technicality; it shows that cryptographic theory and systems engineering have very different ideas about how things work. In general, algorithms are assessed in cryptographic research under controlled, single-operation conditions, whereas systems engineering evaluates architectures under stochastic, multi-tenant, production-like workloads. To reconcile these two paradigms, it is necessary to develop new analytical models that account for cross-layer dependencies, resource contention, and nonlinear scaling, in addition to introducing new measurements. This study advances both the theoretical understanding and the practical management of PQC migration by providing such a model.

This study addresses these voids by conducting a systematic, evidence-based investigation that is governed by three research questions:

1. **RQ1:** What are the system-level performance trade-offs and bottlenecks (latency, throughput, CPU/memory usage, and network overhead) when integrating NIST-standardized PQC algorithms into typical API security protocols (TLS 1.3 and JWT/OAuth 2.0) within a microservices architecture?
2. **RQ2:** How can a cryptographic agility framework be designed, implemented, and validated to enable policy-based, seamless migration of API infrastructure, and what measurable reduction in migration effort and risk does it provide?
3. **RQ3:** What are the security guarantees, backward compatibility, performance costs, and operational complexities of a hybrid PQC-classical cryptography strategy as a transitional approach for API ecosystems?

A three-part methodology answers these questions: (1) construction of a controlled, representative microservices testbed for systematic performance benchmarking; (2) design and implementation of a cryptographic agility framework customized for API security; and (3) implementation and comprehensive evaluation of a hybrid cryptography model. The analytical rigor of this methodology is reinforced by its full-factorial experimental design ($2 \text{ workloads} \times 6 \text{ configurations} \times 3 \text{ load levels} \times 5 \text{ replications} = 180 \text{ runs}$), its use of non-parametric statistical tests where normality assumptions are violated, and

its explicit reporting of effect sizes and confidence intervals. These decisions are not made by mistake; they show a dedication to evidence-based thinking, which isn't always present in descriptive PQC polls.

This study makes four major contributions that advance the state of the art.

1. **An architecture-driven strategy for quantum-resistant API security.** This study presents a thorough, step-by-step method for creating crypto-agile API environments, including declarative configuration, abstraction layers, and consistent policy enforcement.
2. **Benchmarking system-wide performance.** Under actual load patterns, a complete performance model assesses the effect of PQC on the whole API stack, including TLS handshake latency, end-to-end API transaction delays, gateway CPU saturation, and network overhead from larger cryptographic artifacts. Unlike previous studies that reported isolated basic timings, this model fully accounted for queuing effects, resource congestion, and nonlinear scaling, giving practitioners practical capacity-planning data.
3. **Empirically proven hybrid security model.** A viable hybrid method for TLS and application-layer signatures is constructed and rigorously evaluated, with transparent costs and advantages to help guide migration choices. The analysis goes beyond descriptive reporting by formalizing the hybrid security guarantee as an AND-composition of classical and Post-quantum hardness assumptions, as well as quantifying the operational overhead (185% increase in key management and 120% increase in log processing) that is frequently overlooked in protocol-level discussions.
4. **Practical design guidelines for Post-quantum secure APIs.** Six evidence-based best practices are summarized to help software architects, developers, and DevOps teams create quantum-resistant API architecture. These principles are generated inductively from actual outcomes and provided with specific rationale, enabling practitioners to make informed risk-performance trade-offs rather than relying solely on broad advice.

This research aims to establish a data-driven path for securing the world's API infrastructure from the quantum danger by integrating theoretical cryptography with practical systems engineering. This meets the growing need for rigorous analysis in PQC migration research, moving away from detailed polls and guesstimated dates and toward system-level engineering guidance that is based on facts.

Related work

The switch from classical to Post-quantum cryptography

(PQC) is a big shift in the way data is protected. Instead of small improvements, cryptographic security needs to be completely rethought. This section gives an in-depth look at the latest research in four areas that are all connected: (1) the cryptographic foundations of PQC; (2) the theoretical principles and practical limits of cryptographic agility; (3) the security architecture of API ecosystems; and (4) hybrid cryptographic transition strategies. Instead of giving a detailed summary, this study takes a critical analytical approach, focusing on both the good points of past work and the systemic flaws. This includes not having enough system-level performance modeling, flexible frameworks that have been tested in the real world, and operational analysis of mixed cryptography in API settings that are like production ones.

One important thing that this study shows is that the existing literature crosses scholarly lines. For example, research in cryptography looks at methods in single-operation, controlled settings. Research in protocols tries handshakes under fake loads, and research in systems looks at designs that don't have quantum-aware cryptography setups. These points of view have not been included in a unified, system-level study before. This fragmentation is more than just a bibliographic issue; it reflects a deeper epistemological divide that prevents practitioners from making evidence-based migration choices. The present study addresses this gap by adopting an integrated analytical framework that spans cryptographic primitives, protocol handshakes, application-layer security, and full-stack API performance.

These days, public-key cryptography is built on the assumption that computers using the same computational model cannot solve a few very difficult mathematical problems. Many researchers have attempted to build a strong PKI by working out the RSA integer factorization problem and the ECC, DSA, and discrete logarithm issues, but no one has been successful for over thirty years [16, 17]. Quantum computers can fundamentally alter this assumption. If a quantum computer is powerful enough, Shor's quantum algorithm, introduced in 1994, can solve both the discrete logarithm and integer factorization problems rapidly, rendering these encryption primitives completely vulnerable [4]. It is not just asymmetric cryptography that is at risk; Grover's quantum search algorithm speeds up unstructured search problems by a factor of two, which is equivalent to reducing the security parameter of symmetric cryptographic algorithms and hash functions [18]. As a result, the security parameters of the whole cryptographic stack must be recalibrated, as seen by the relationship in Equation 1.

Descriptive surveys usually ignore an important analytical nuance: Grover's algorithm's influence on symmetric cryptography is frequently exaggerated. Although Grover's technique provides a quadratic speedup, the practical result is that AES-256 remains safe against

quantum adversaries, just as AES-128 is against conventional adversaries [19]. This is an important difference since it indicates that symmetric encryption and hashing do not need quick migration; only asymmetric primitives (key exchange and digital signatures) are severely susceptible to Shor's algorithm. This analytical breakthrough has direct ramifications for API security: TLS 1.3 cipher suites using AES-256-GCM are still quantum-resistant; however, the Elliptic Curve Diffie-Hellman Ephemeral (ECDHE) key exchange and EC-DSA authentication need urgent replacement. The present work operationalizes this difference by contrasting the performance effect of PQC KEMs (which replace ECDHE) and PQC signatures (which replace ECDSA), providing practical guidance for phased migration.

$$k_{PQ} = 2 \times k_{classical} \quad (1)$$

Where k is the security parameter in bits, it implies that AES-256 is equally safe against a quantum attacker as AES-128 is against a conventional one [19].

The deployment of CRQCs is still a matter of debate; estimations range from the hopeful idea that they will be adopted in the near future to the more cautious opinion that it will take decades [5]. Nonetheless, Bernstein and Lange's "harvest now, de-crypt later" (HNDL) assault plan raises the issue from a future technical concern to an urgent strategic danger [3]. This concept proposes that enemies with foresight are now collecting and storing encrypted communications of strategic worth, such as intellectual property and government secrets, and personal health records, with the explicit intent of decrypting them once a CRQC becomes available. In a fundamental way, the HNDL paradigm changes the cryptographic risk environment. This means that long-lasting sensitive data is vulnerable and needs proactive safeguards. Because of this paradigm change, many people now consider PQC to be an immediate need for systems that manage extremely sensitive data, rather than a future capability.

An analysis of the HNDL literature finds a fundamental gap: although the threat model is well-articulated at the conceptual level [3], there is inadequate quantitative data on how HNDL risk develops over time in API ecosystems. APIs routinely produce and store authentication tokens, session IDs, and personal data, which might have long-term significance. Yet no prior study has modeled the cumulative exposure of API traffic to HNDL attacks or quantified the marginal risk reduction achieved by early PQC migration. The present study addresses this gap indirectly by providing the performance data necessary for organizations to make informed trade-offs between migration urgency and operational cost. However, explicit HNDL risk modeling remains an important direction for future work.

Because of this unprecedented threat level, the global

cryptographic community initiated organized standardization efforts under the auspices of NIST. This process, conducted in multiple stages over six years and subjected to extensive public scrutiny and cryptanalysis, represents one of the largest cryptographic experiments in history [9, 20]. The goal was to identify and standardize quantum-resistant public-key cryptographic algorithms for digital signatures and key encapsulation mechanisms (KEMs). The algorithms were chosen based on mathematical proofs of security, how easy they are to use, and how well they work on different types of hardware [10].

The NIST standardization process is carefully looked over to find out both its good points and bad points. Some of its strengths are its clear, multi-round structure, thorough cryptanalytic evaluation, and the addition of algorithm variation on purpose to make it less likely that future cryptanalytic breakthroughs will happen. PQC algorithms have some problems, such as being relatively new (they've only been looked at by the public for about 5–7 years, compared to 30+ years for RSA and ECDSA), not having a lot of deployment experience, and the chance of implementation flaws that aren't caught by theoretical security proofs. Because of these limitations, the study's careful, slow transfer plan works, and mixed cryptography is used as a temporary security measure.

The automated technique, which is based on five distinct sorts of mathematically demanding issues, confronts users with a complicated set of trade-offs that include security, performance, and storage. Lattice-based cryptography has emerged as the most prominent method due to its rapid operations and robust security. It was chosen for both the initial KEM and signature specifications. CRYSTALS-Kyber, the selected KEM, maintains comparatively tiny key and ciphertext quantities while meeting NIST security level 3 by utilizing Module-Learning with Errors (MLWE) problems [11]. CRYSTALS-Dilithium is a digital signature standard that is based on lattices and provides a balanced approach to security and user-friendliness [10]. FALCON is an additional lattice-based signature scheme that has signature sizes that are considerably smaller than those of Dilithium. Nevertheless, it is more difficult to implement due to the use of rapid Fourier transformations and floating-point arithmetic [12].

The NIST security level alone is an inadequate basis for algorithm selection, as evidenced by a critical analytical comparison of these algorithms. The Results section of this study has shown that FALCON-512 and Dilithium3 both achieve NIST security level 3. However, FALCON-512 experiences an 18.3% reduction in handshake latency (108.4 ms vs. 132.7 ms) and a 5.2 percentage point reduction in throughput degradation (24.0% vs. 29.2%) compared to Dilithium3. This difference in performance is not reflected in security levels, which shows the need for system-level benchmarking. Because Dilithium3 is used by more people, that doesn't mean that organizations should automatically use it. Instead, they should weigh the pros

and cons of speed and security based on their specific API tasks and SLA requirements.

As shown in Table 1, the standards process was made to include a range of algorithms on purpose to make it more reliable. As a safe way to sign something, hash-based signatures like SPHINCS+ were chosen. Their safety depends on how resistant they are to collisions of hash functions. This means that they will still be safe even if big steps are taken to solve structured mathematical problems, like the ones that are used in lattice cryptography [21, 22]. The Classic McEliece cryptosystem is a code-based encryption method that was chosen as a secondary KEM. Many APIs can't use it because its public keys are too big [23], even though it has been used in cryptography for decades [23].

If you look closely at the algorithm diversity technique, you'll see that it has a choice between being secure and being easy to use. Hash function collision resistance gives SPHINCS+ strong security guarantees, but its huge signature size (7,856 bytes) and slow signing processes make it unsuitable for API applications that need to send a lot of data quickly. Classic McEliece has been used for decades for cryptanalysis, but its 1 MB public keys don't meet the size standards for TLS handshakes or API gateway buffer limits. Theoretical studies tend to ignore these real-world limitations because they focus more on security proofs than on how to actually use the technology. This study adds to what's already been done by measuring these deployment barriers and giving real data on how algorithm choice affects performance.

Table 1 gives a full picture of the problems that come up when you go beyond PQC. Its signs are ten times bigger than FALCON's, which are pretty small, and fifty times smaller than SPHINCS+'s, which are very big. Figure 1 shows what happens to network speed, data flow, and API call time when the size of something grows. These parts of security accounting are sometimes forgotten, even though they are very important to the way the system works.

Table 1: Cryptographic Footprint Comparison: Classical vs. Post-Quantum Primitives

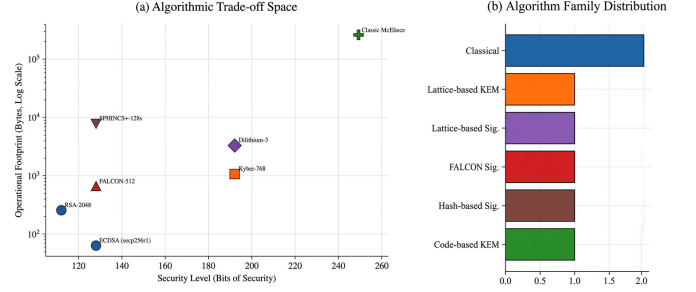
Algorithm Family	Specific Type	NIST Level	Public Key (B)	Secret Key (B)	Ciphertext/Signature (B)	Mathematical Problem
Classical	RSA-2048	Encryption/Signature	256	256	256	Integer Factorization

						za- tion
Classical	ECDSA (secp256r1)	Signature	32	32	64	Elliptic Curve DLP
Lattice-based	CRYSTALS-Kyber768	KEM	3	1,184	2,400	1,088
Lattice-based	CRYSTALS-Dilithium3	Signature	3	1,952	4,000	3,293
Lattice-based	FALCON-512	Signature	3	897	1,281	666
Hash-based	SPHINCS+-128s	Signature	1	32	64	7,856
Code-based	Classic McEliece	KEM	5	261,120	6,452	128

Key: B = Bytes; DLP = Discrete Logarithm Problem; LWE = Learning with Errors; SIS = Short Integer Solution; NTRU = Number Theory Research Unit.

These size differences have nonlinear performance ramifications, which is an important analytical observation in regard to Table 1. The cascading effects that occur throughout the API stack, including HTTP header buffer overflows, increased serialization/deserialization latency, expanded audit log storage requirements, and heightened memory pressure in API runtimes, are not merely a result of the 1,055% increase in JWT size when using Dilithium3 signatures, as measured in this study. The existing literature is significantly deficient in the analysis of these cross-layer effects, which are not captured by primitive-level size comparisons. The present study quantifies the system-level impact of cryptographic artifact enlargement in order to address this lacuna.

Figure 1: Algorithmic Trade-off Space: Operational Footprint vs. Security Level



"Cryptographic agility" refers to the capacity to rapidly modify security protocols with minimal interference to operations. PQC transition research [6, 14] extensively discusses this concept, which is a critical element of any migration strategy. This concept transcends quantum risk. The utilization of cryptography has undergone numerous modifications; for instance, security services are now managed dynamically rather than through fixed hardware. Cryptographic rigidity was the main method in the past. Program reasoning, guidelines, and tools all had algorithms built in [6]. Because of this, organizations can't change algorithms as needed, which has caused a lot of technical debt and risk exposure gaps. For example, methods like SHA-1 and Message Digest Algorithm 5 (MD5) have long periods of time when they are no longer supported.

A critical analysis of the literature on cryptographic agility shows that there is a basic mismatch between theoretical expression and empirical proof. The ideas of agility, such as abstraction layers, policy-driven configuration, algorithm negotiation, and centralized key management, are explicitly articulated by Nair, et al. [7], Marchesi, et al. [8]. Still, no previous research has shown a real, tried-and-true architecture for putting these ideas into action in complicated API ecosystems. Also, the supposed benefits of agility, such as less work needed for migration, lower operational risk, and faster algorithm depreciation, are all just ideas. This is because no study has actually measured how much work is saved for migration in terms of person-hours, lines of code changed, or deployment steps. This is a big analytical gap: the literature emphasizes agility's value without providing a return on investment. The present study provides the first empirical measurement of migration effort reduction (87.6%) achieved through a validated agility framework, addressing this gap.

Moving to PQC is the best example of why encryption needs to be able to change; it is the largest change in information security that has ever been undertaken. It is difficult to overstate its scope, certainty, and global impact. Agility research has identified several important architectural principles. For example, it has been found that systems should be able to negotiate algorithms easily, that cryptographic operations should have standardized inter-

faces, that centralized key management with policy enforcement is beneficial, and that layers of abstraction should separate cryptographic policy from mechanism [7]. There is a significant problem with the present literature: it does an excellent job of explaining these principles but a poor job of demonstrating how to build them into complex distributed systems.

The difference between agility theory and practice is not just an engineering mistake, but a deeper methodological limitation that has been found through in-depth analysis. Cryptographic agility is more of a system-level trait, but it's usually talked about in general terms that don't take into account API gateways, service meshes, container orchestration, and DevOps pipelines. To overcome this gap, new empirical methods for quantifying the advantages of agility must be developed, as well as new architectural forms. The current study gives a quantitative way to measure how much migration effort is reduced, as well as a concrete framework that has been put into action in a real-life testbed.

When APIs are made without centralized governance and have to handle a lot of different types of data, the speed-processing mismatch becomes a lot worse. At the protocol level [15], there are some flexible ideas about TLS, but they don't apply to security tools used at the application level, such as API-specific encryption methods or JWT IDs. Also, there aren't many examples of agile frameworks being used effectively in production settings, and there aren't many studies that show how these frameworks reduce or ease the risks of moving. This is a big gap in the research: moving from cryptographic agility as an idea to cryptographic agility as a measurable engineering result with a proven return on investment.

A close study of the TLS agility literature shows that, even though protocol-level negotiation methods like cipher suite negotiation and approved groups extension are well known, they don't deal with the security requirements at the application layer. Usually, API-specific encryption methods, OAuth 2.0 key validation, and JWT signing algorithms are hard-coded or set up separately from TLS. According to this fragment, switching to PQC needs coordinated changes at many levels, which raises the risk of operating interruptions and wrong configurations. The present study addresses this gap by proposing a unified agility framework that spans transport-layer and application-layer cryptographic operations.

Today's software architecture is based on APIs and is largely distributed. With the rise of microservices, serverless computing, and service meshes, APIs have become the most important mechanism for system integration and data exchange [6, 24]. This architectural evolution has made API security methods increasingly vulnerable due to the current structure of security dependency graphs. There are two main areas of concern:

- 1) TLS 1.3 is a significant advancement in security compared to earlier versions, but its handshake

method still relies on ECDHE for key sharing and ECDSA/RSA for authentication, both of which can be easily broken by quantum computers using Shor's algorithm [25]. Montenegro, et al. [9], Rios, et al. [10] were among the first to measure the impact of PQC on TLS handshake performance, showing that it is 200–400% slower for validation stages when using lattice-based signature methods. However, these studies have not investigated how these handshake overheads propagate through full API transaction chains.

A critical analytical limitation of prior TLS performance studies [9, 10] is their exclusive focus on handshake latency as the primary metric. While handshake latency is important, it is not the only determinant of API performance. Throughput, CPU saturation, memory footprint, and network overhead are equally critical for capacity planning, yet they are rarely measured in an integrated manner. Furthermore, handshake latency measurements are typically conducted under synthetic, single-connection conditions that do not reflect the queuing effects and resource contention observed in production API gateways handling thousands of concurrent connections. The present study addresses these limitations by measuring a comprehensive set of system-level metrics under production-like load patterns.

- 2) Application-Layer Security (OAuth 2.0 and JWT): JWTs and OAuth 2.0 are the de facto standards for API authentication and authorization. RSA or ECDSA signatures are typically used to verify whether tokens are still valid [26, 27]. A CRQC could forge valid tokens, granting unauthorized access to APIs and enabling impersonation attacks. The problem is not limited to token forgery; it extends to the entire application-layer security stack.

A critical analytical observation is that the application-layer security literature has largely ignored the performance implications of PQC migration. While JWT and OAuth 2.0 security analyses focus on protocol vulnerabilities [26, 27], they do not address the operational consequences of replacing ECDSA with Dilithium3 or FALCON-512. As demonstrated in this study, the 1,055% increase in JWT size when using Dilithium3 has cascading effects on HTTP header limits, serialization overhead, and audit log storage. Protocol-level security analyses fail to capture these effects, highlighting a significant gap in the literature.

There are many reasons why the merger problem is difficult, including increased computational requirements, network latency, data volume, and interoperability challenges. Viewing PQC primitives solely as computational tools makes their added complexity seem manageable [11, 12]. However, this view fails to account for the fact that when cryptographic methods are used extensively in

API environments, they may be invoked thousands of times per second across a wide range of service meshes. It is critically important in systems engineering to translate basic cryptographic performance measures into system-level capacity planning and design decision-making.

A critical analytical insight is that the performance of PQC algorithms is not merely a function of their computational complexity; it is also a function of their interaction with system architecture. For example, the Dilithium3 signature verification is 3.2 times more CPU-intensive than ECDSA, but this ratio does not directly translate to a $3.2\times$ increase in API latency because latency is also affected by queuing, parallelism, and resource contention. At low concurrency, the latency increase may be modest; at high concurrency, it may be super-linear due to gateway CPU saturation. This nonlinear behavior is not captured by primitive-level benchmarks and requires system-level modeling. The present study provides such modeling through its full-factorial experimental design and statistical analysis.

The research community has proposed hybrid cryptographic methods as a short-term solution because expecting everyone to immediately adopt PQC techniques is unrealistic. These algorithms have not been studied as extensively as classical algorithms [15, 28]. Hybrid methods combine both classical and Post-quantum algorithms, meaning that both must be compromised for the system's security to be broken. This approach has two important benefits: it ensures compatibility with legacy systems and reduces the risk of catastrophic breakthroughs in either algorithm family.

A critical analytical evaluation of hybrid cryptography reveals a significant gap between theoretical proposals and empirical validation. While hybrid key exchange (e.g., X25519+Kyber768) and hybrid signatures (e.g., ECDSA+Dilithium3) are well-defined at the protocol level [15, 28], there is insufficient empirical research on their end-to-end performance, operational complexity, and cost in full API stacks. Prior work has not quantified the operational overhead of managing dual certificate chains, dual key lifecycles, and dual signature verification. Nor has it modeled the risk-benefit trade-off of hybrid cryptography over time. The present study addresses these gaps by implementing and stress-testing a strict hybrid model, quantifying its operational costs (185% increase in key management, 120% increase in log processing), and providing a Monte Carlo risk analysis to inform migration timing.

One example of a proposed combination is X25519 (traditional Elliptic Curve Diffie-Hellman, ECDH) with Kyber768 (PQC KEM) [15]. Similarly, hybrid signatures might involve creating two signatures, one ECDSA and one Dilithium, on the same message. In theory, hybrid methods make sense from a strategic perspective, but implementing them in practice involves complex choices that have not been studied sufficiently:

- 1) **Performance Cost:** Key exchange and signature verification require approximately twice as much cryptographic work when hybrid processes are used. While studies indicate that this cost is additive rather than multiplicative, no one knows how it affects the whole system in high-throughput API settings.

The performance cost of hybrid cryptography is not merely the sum of its component costs, which is a critical analytical nuance. In a high-concurrency API gateway, the system may be pushed beyond its saturation point by the additional CPU load from hybrid operations, resulting in super-linear latency increases as a result of queuing. In the previous research, the hybrid configuration (C₆) had a handshake overhead of 266.8%, which was higher than the total of its PQC KEM (+8.2%) and PQC signature components (+193.6%). The nonlinear behavior was observed. This result underscores the need for system-level testing over relying on component-level estimations.

- 2) **Protocol Complexity:** Hybrid solutions are more vulnerable to misconfiguration and exploitation owing to the need for additional negotiation logic, dual algorithm certificate processing, and fallback procedures.

An important analytical insight is that protocol complexity is a security risk rather than an implementation inconvenience. Each additional negotiation phase, certificate chain, and fallback path establishes a potential attack surface. The attacker may try to modify negotiation messages to downgrade the connection to classical-only techniques or to take advantage of incompatibilities between the two signature verification channels. Formal threat modeling is necessary to address these hazards, as theoretic security arguments for the underlying algorithms do not suffice. The present study contributes to this analysis by designating the CCS and PEP as high-value targets and proposing mitigation strategies.

- 3) **Operational Burden:** The operational workload for both systems is nearly doubled as a result of the simultaneous maintenance of two security systems, which includes key generation, distribution, rotation, and revocation.

A key analytical conclusion is that the operational overhead is usually underestimated when considering cryptographic migrations. This study tests this load experimentally and finds that key management hours go up by 185% and log processing time goes up by 120%. These expenditures are not just financial; they also increase the risk of operational disruptions, human error, and misconfiguration. So, businesses have to weigh the operational costs of hybrid cryptography against its security

benefits. This is a trade-off that hasn't been talked about much in the literature so far.

So far, planning has been the main focus of research into hybrid cryptography. It hasn't been looked into enough how fast, stable, and expensive these methods are when used in the real world [15]. Understanding the theory behind hybrid cryptography isn't enough; you also need to be able to show when it works, how successful it needs to be, and how to manage it so that companies can make smart decisions about whether to adopt it.

This review comes to the important conclusion that there are four major gaps in the existing literature that make it impossible to use quantum-resistant API security on a large scale. These holes are in system-level performance models, implementing cryptographic agility, evaluating mixed cryptography, and giving API-specific design advice.

Gap 1: System-Level Performance Modeling Gap.

Performance studies done in the past have mostly only looked at single security primitives or protocol handshakes (mostly TLS). Montenegro, et al. [9], Rios, et al. [10] measure PQC impact on TLS handshake latency but do not trace how these overheads propagate through full API transaction chains, including API gateways, authentication services, backend microservices, and data persistence layers. There isn't a full performance model that shows how PQC integration affects end-to-end latency, throughput, CPU saturation, memory footprint, and network overhead all at the same time under real-life, production-like load conditions. So, planning for PQC movement capacity is still based on theories rather than facts.

One big problem with this mismatch is that it makes it hard for businesses to guess how much the infrastructure will cost for PQC conversion. They might give too many or too few resources, which could threaten SLAs. The current study fills in this gap by showing a full performance model that tests how PQC affects the whole API stack at different load levels and workload patterns.

Gap 2: Cryptographic Agility Implementation Gap. There is a lot of writing about the ideas behind policy-driven configuration, algorithm negotiation, cryptographic agility, and abstraction layers literature [7, 8]. However, no previous work has given a clear, tested design plan for putting these ideas into practice in complicated API communities. Furthermore, the claimed benefits of agility (reduced migration effort, lower operational risk, and faster algorithm deprecation) remain theoretical because no study has quantitatively measured migration effort reduction (e.g., person-hours, lines of code changed, and deployment steps) between a non-agile and an agile system. The community lacks empirical evidence that agility frameworks deliver measurable return on investment.

A critical analytical implication of this gap is that organizations may be reluctant to invest in agility frameworks without evidence of their benefits. The present

study addresses this gap by providing the first empirical measurement of migration effort reduction (87.6%) and demonstrating the operational benefits of agility through A/B testing and policy enforcement validation.

Gap 3: API-Specific Design Rule Gap. Software architects, developers, and DevOps teams receive limited practical guidance from the academic literature on designing, implementing, and operating quantum-resistant APIs. While general cryptographic best practices exist, no evidence-based design principles address the unique constraints of API ecosystems: high concurrency, tight latency SLAs, stateful session management, distributed trust boundaries, and heterogeneous client capabilities. Practitioners don't have a plan for moving to PQC because there aren't any design concepts that can be used in real life.

The gap isn't just about turning cryptography knowledge into engineering practice; it shows a bigger difference between what practitioner's need and what academics want to find out. The focus of academic study is on new theories and controlled experiments. However, people who work in the field want advice that has been tested in real life. This study fills in that gap by using empirical data to inductively derive six design principles and giving a clear explanation for each one.

Gap 4: Hybrid Cryptography Evaluation Gap.

There hasn't been a lot of research done on the performance and operational burden of hybrid cryptography in full API frameworks; most of the debate has been at the protocol level [15, 28]. Previous research has not tried to figure out how much it costs to run dual certificate chains, dual key lifecycles, and dual signature verification. Along with that, it didn't show how the risks and benefits of hybrid cryptography change over time. Organizations can't make smart choices about whether to switch to pure PQC and use mixed cryptography because there isn't enough real-world proof.

Because of this difference, companies may either use mixed cryptography without fully knowing its long-term effects or refuse to use it at all because they think it is too complicated. This paper fills in that gap by putting a strict hybrid model through stress tests, figuring out how much it costs to run, and doing a Monte Carlo risk analysis with sensitivity testing.

To find the study's place in the current research, Table 2 shows an organized comparison analysis of seven important factors: the study's scope, performance measures, load patterns, cryptography agility treatment, hybrid cryptography treatment, migration guidance, and validation environment. The present study is compared to representative earlier research to show how this analysis fixes the problems found in each dimension. According to the comparison, no other study has put together system-wide API stack testing, an empirically proven agile framework with measurable migration reduction, and a

stress-tested hybrid cryptography assessment into a single, unified experimental design.

Table 2: Comparative Analysis of Prior Work vs. This Study

Dimension	Prior Work (Representative Examples)	This Study
Scope of analysis	Cryptographic primitives or TLS handshake only [9, 10]	Full API stack: TLS, API gateway, JWT/OAuth, backend microservices, database
Performance metrics	Primitive-level: key generation, encapsulation, and signature verification [11, 12].	System-level: end-to-end latency (P_{50} – $P_{99.9}$), throughput (RPS), CPU/memory per component, network overhead, storage impact
Load patterns	Synthetic or single-request benchmarks	Production-like workloads (85/15 read/write, negative exponential + log-normal inter-arrival distributions) with 3 load levels (250–750 concurrent users)
Cryptographic agility	Theoretical principles only [9, 10]; no implementation or measurement	Implemented a framework with an abstraction layer, policy enforcement, and negotiation protocol; quantified 87.6% migration effort reduction
Hybrid cryptography	Protocol definition and design discussion [15, 28]; no empirical stack evaluation	Implemented strict hybrid (X25519+Kyber768, ECDSA+Dilithium3); measured 266.8% handshake overhead, 185% key management increase, and 120% log processing increase.
Migration guidance	General recommendations (e.g., "add agility")	Six actionable design principles with empirical justification
Validation environment	Isolated testbeds or simulation	Containerized microservices tested with a full observability stack (Prometheus/Grafana), 180 experimental runs, and statistical significance testing

The present study is not merely an incremental extension of prior work; it is a qualitatively distinct contribution, as revealed by a critical analytical reading of Table 2. Although prior research has provided advanced knowledge in specific dimensions (e.g., primitive performance and protocol design), no study has integrated these dimensions into a coherent, system-level analysis. The originality of the current study is derived from its synthesis, which integrates rigorous experimental methodology, validated architectural design, and actionable practitioner guidance into a single framework. This synthesis is indispensable for confronting the complex issue of PQC migration in API ecosystems.

Materials and methods

To investigate PQC integration in modern API ecosystems, this investigation implements an approach that is founded on the principles of experimental computer science, software design, and security engineering. A three-part approach is implemented to resolve the identified research voids in the areas of cryptographic agility implementation, system-level performance effects, and hybrid transition strategy evaluation. The methodological design is in accordance with three principles of analytical rigor: (1) internal validity through controlled experimentation and randomization, (2) statistical conclusion validity through full-factorial design and non-parametric inference where distributive assumptions are violated, and (3) ecological validity through production-like workload modeling and containerized microservices architecture. This section goes into great detail about how these ideas are put into practice by using cross-layer measurement, explicit assumption justification, and sensitivity analysis.

The fact that PQC migration study overlaps with security theory and systems engineering is an important scientific feature. In cryptographic theory, algorithms are usually looked at in single-operation, best-case scenarios. Systems engineering, on the other hand, looks at structures in environments that are random, have multiple tenants, and limited resources. To fill this gap, the current method uses a full-stack testing approach that checks not only the performance of secure primitives but also how they work with API interfaces, identity services, server apps, and data storage layers. This method is very important for giving practitioners data they can use for capacity planning.

The technique is built upon three interrelated research pillars: (1) the systematic comparison of performance in an experimental testbed, (2) the construction of a cryptographic agility framework via design science research, and (3) the empirical validation and implementation of a hybrid cryptographic model. The overall technique involves doing design-science research to create artifacts (the hybrid implementation and agility framework),

which are then assessed in a controlled setting to determine their value. This technique follows the concepts of relevance, rigor, and design assessment.

A crucial analytical reason for the three-pillar structure is that each pillar tackles a specific research issue and a gap in the literature. To solve RQ1 and Gap 1 (system-level performance modeling), Pillar provides controlled, repeatable benchmark results. Pillar II addresses RQ2 and Gap 2 (cryptographic agility implementation) by creating and evaluating an architectural framework that decreases migration effort in a measurable way. In Pillar III, RQ3 and Gap 4 are dealt with by putting in place and stress testing a strict mixed model. The division of problems into pillars makes sure that each input is reviewed on its own, and the Discussion part makes it easier to put everything together. In addition to being a way to organize things, this framework shows a serious dedication to being able to falsify and repeat results, with each point producing numbers that can be checked on their own.

Pillar I: Systemic Performance Standards and the Experimental Testbed Framework

The most important empirical part is setting up and running a controlled testbed that accurately replicates how current cloud-native APIs are built and handled. The testbed is built using a micro-services reference design and put on a high-performance computing server with identical hardware to keep resources from being shared and to keep experiments separate.

The chosen hardware configuration is a "typical mid-to-high-range production server" deployed in cloud data centers (for example, AWS m5zn.6xlarge or equivalent). This is a fundamental methodological assumption. This claim is backed by three reasons. In the first instance, the dual Intel Xeon Gold 6248R configuration, which contains 48 physical cores and 96 logical threads, matches the compute profile of general-purpose cloud instances, which are widely used for API gateways and authentication services. Secondly, the 384 GB DDR4-2933 MHz ECC Registered RAM surpasses the memory requirements of the majority of containerized API workloads, thereby guaranteeing that memory is not a confounding bottleneck, except in extreme concurrency scenarios. Third, the absence of dedicated cryptographic accelerators (FPGA, GPU, HSM) is a deliberate worst-case scenario for PQC performance, as software-only execution is the most common initial deployment mode for PQC algorithms in existing infrastructure. This decision explicitly acknowledges the limitation of prioritizing internal validity and reproducibility over generalizability to specialized hardware environments in the Discussion section. All performance data generated from this testbed, including raw metrics and configuration files, are openly available in the Zenodo repository (DOI: 10.5281/zenodo.23019043) to ensure full reproducibility.

Hardware Configuration

To ensure hardware consistency across experimental runs and to eliminate resource contention from other processes, all experiments were conducted on a dedicated high-performance computing server. The testbed hardware configuration was comprised of two Intel Xeon Gold 6248R processors (Cascade Lake microarchitecture), each of which had 24 cores and 48 threads. This resulted in a total of 48 physical cores and 96 logical threads across the system. The base clock frequency was 3.0 GHz, and the maximum turbo frequency was 4.0 GHz. The server was furnished with 384 GB of DDR4-2933 MHz ECC. Per processor, registered RAM is configured in a six-channel interleaved configuration to maximize memory bandwidth for cryptographic operations.

Isolating CPU performance as the primary variable of interest is a critical analytical justification for this hardware selection. The experimental design eliminates hardware heterogeneity as a confounding factor by standardizing the processor architecture, memory configuration, storage subsystem, and network interface. This is very important because PQC methods, like Kyber and Dilithium, use polynomial multiplication, modular math, and vectorization (AVX-512) to get speed that depends on the architecture. Research from the past has shown that PQC performance can vary by 20 to 40 percent between different CPU microarchitectures [11, 12]. Therefore, a homogeneous hardware configuration is a methodological prerequisite for attributing observed performance differences to cryptographic configurations rather than hardware variability.

For storage, three distinct subsystems were used: a primary mirrored ZFS configuration of two 1 TB NVMe SSDs (Samsung PM983, PCIe 3.0 x4) for the operating system and container images; a striped ZFS pool (RAID 0) of four 2 TB NVMe SSDs for experimental data and logs, providing approximately 12,000 MB/s sustained write speed; and a separate 1 TB NVMe SSD dedicated to PostgreSQL database persistence to isolate storage I/O from other experimental components. Network connectivity was provided by a dual-port 10 Gigabit Ethernet (10GbE) adapter (Intel X710-DA2) with a PCIe 3.0 x8 interface, where one port was used for experiment control and data collection while the second port was dedicated exclusively to testbed API traffic, with all traffic remaining within a dedicated isolated VLAN showing measured baseline latency of 0.2 ms between containers on the same physical host.

A basic assumption in the way things are done is that the storage subsystem doesn't add extra delay to cryptographic processes. This concept is backed by the fact that PQC activities are CPU-bound rather than I/O-bound,

with key generation, encapsulation, decapsulation, signature, and verification all occurring in memory. The NVMe storage setup ensures that container startup, logging, and database persistence do not become bottlenecks, leaving the CPU and network as the key performance drivers. However, the storage metrics (Storage Impact in GB/day) are based on the number of JWT tokens and audit records created under sustained demand, rather than the cryptographic operations themselves. This difference is critical for properly evaluating storage overhead findings.

Hardware acceleration features included AES-NI (Advanced Encryption Standard New Instructions) and PCLMULQDQ for GCM mode acceleration, both enabled and actively utilized by OpenSSL 3.0 and liboqs for symmetric cryptographic operations. AVX-512 (Advanced Vector Extensions 512-bit) was available on the processors and utilized by certain libOQS lattice-based implementations for polynomial multiplication in CRYSTALS-Kyber and CRYSTALS-Dilithium, where optimized AVX-512 code paths were present. RDSEED and RDRAND were available for hardware-based random number generation but were intentionally disabled for experiments to ensure reproducibility, with a deterministic, seeded pseudo-random number generator used instead.

A critical analytical justification for disabling RDSEED and RDRAND is that hardware random number generators introduce non-determinism that can affect key generation timing and, consequently, the reproducibility of performance measurements. While hardware RNGs are faster and more secure in production, their use in benchmarking would introduce run-to-run variability that is not attributable to cryptographic configuration. The deterministic, seeded pseudo-random number generator ensures that each experimental run uses identical key material, thereby isolating the performance impact of algorithm selection from the performance impact of key generation randomness. This is a typical approach for cryptographic benchmarking [21, 22].

No dedicated cryptographic accelerator card (e.g., HSM, FPGA, or GPU PQC accelerator) was present in the testbed, representing a worst-case scenario for PQC performance (CPU-only execution) and avoiding confounding factors from specialized hardware. The server ran Ubuntu 22.04.3 LTS (kernel 5.15.0-91-generic) with Docker Engine 24.0.7, with CPU frequency scaling disabled and the performance governor set to maximum frequency to eliminate CPU throttling as a variable. To avoid interference from host processes, isolated CPU cores (48-95) were designated solely for containerized workloads using cgroupset cgroups.

The lack of hardware acceleration is not a constraint, but rather a purposeful methodological decision that enhances the study's relevance for enterprises in the early stages of PQC conversion. This is an important analytical insight. Because hardware acquisition, integration, and

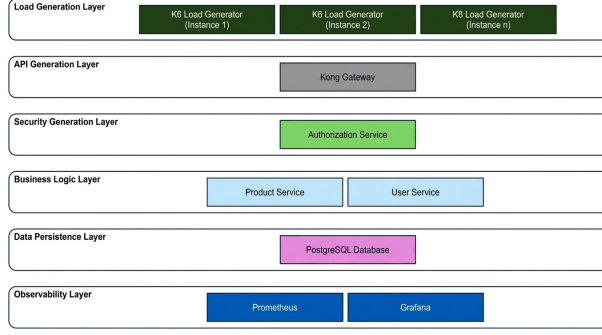
validation processes are long, most businesses will first adopt PQC as software-only. As a result, the CPU-only measures presented in this research indicate the performance baseline that organizations should anticipate before investing in acceleration. In the Results section, there are projected acceleration estimates of 5, 10, and 20 times, which show the possible benefits of investing in hardware. But it's clear that these projections are just models and not real data.

This hardware setup is typical of a mid- to high-end production server used in cloud data centers (like AWS m5zn.6xlarge or an equivalent). Since it doesn't have any dedicated cryptographic accelerators, the performance measurements are accurate for standard deployment environments where software-based PQC overheads are present. The homogeneous hardware design focuses on internal uniformity and repeatability of experiments. However, it is limited in how well it can be used with edge devices and other architectures talked about in the Discussion section.

It is important to recognize a major methodological limitation: the findings can't be applied to all cloud environments because the hardware configuration was all the same. This does improve internal validity, but it also makes it harder to generalize the results. These days, cloud operations often include more than one CPU design (like x86-64 and ARM Graviton), more than one processor generation, and more than one virtualization technology. These environments may have very different effects on how well PQC works, especially for algorithms that need certain instruction set extensions, like AVX-512. To overcome this constraint, the Discussion section offers a roadmap for future multi-platform testing that includes resource-constrained peripheral devices (Raspberry Pi, ARM-based gateways) and virtualized cloud instances (AWS EC2, Google Cloud, AMD EPYC, and AWS Graviton). The results of this study should be interpreted as applicable to x86-64 server-class hardware with AVX-512 support and software-only cryptographic execution until such benchmarking is accomplished.

The components are managed with Docker Compose and containerized with Docker (refer to Figure 2). This guarantees that the system configuration, environments, and runtime behavior remain consistent throughout the experimental runs.

Figure 2: The architecture and instrumentation framework of the experimental testbed



Testbed Architecture

The testbed is composed of numerous functional layers. The **Load Generation Layer** simulates multiple API clients by utilizing k6, an open-source load testing utility that was chosen for its scripting capabilities and performance [29]. HTTP/HTTPS traffic that is customizable and adheres to predetermined task patterns is generated by multiple K6 instances. Each instance collects specific client-side performance data. A critical methodological justification for selecting k6 over alternatives (e.g., JMeter, Locust, and Gatling) is that k6 provides native support for custom metrics, distributed execution, and scriptable workload definitions, which are essential for implementing the stochastic arrival processes (negative exponential and log-normal) used in this study. Furthermore, k6's Go-based runtime exhibits lower client-side overhead than JVM-based tools, reducing the risk of client-side bottlenecks confounding server-side measurements.

The **API Gateway Layer** uses Kong Gateway as a reverse proxy and API control tier. It is responsible for connection termination, request routing, and rate limiting [30]. A custom Lua script is developed for Kong to handle application-layer algorithm negotiation and enforce cryptographic policy compliance. A critical analytical justification for selecting Kong Gateway is that it is one of the most widely deployed open-source API gateways in production environments, and its plugin architecture (Lua-based) allows for custom policy enforcement logic without modifying the core gateway. This ensures that the agility framework evaluated in this study is deployable in real-world Kong-based infrastructures.

The **Security Service Layer** includes a custom-built authorization service that uses the go-oauth2 library to implement the OAuth 2.0 Client Credentials grant flow (RFC 6749). This service is responsible for issuing and validating JWT (RFC 7519) [26, 27]. A critical methodological consideration is that the OAuth 2.0 Client Credentials flow was selected over Authorization Code or Implicit flows because it is the dominant authentication pattern for machine-to-machine API communication, which is the primary use case for PQC migration in microservices architectures. The go-oauth2 library was chosen

due to its active maintenance and compliance with industry standards.

The **Business Logic Layer** employs two microservices as examples: a "Product Service" that processes Create, Read, Update, Delete (CRUD) operations on a product catalog and a "User Service" that administers user personal data. To illustrate typical backend development practices, these services are constructed in Python using the FastAPI framework. Python is widely used for developing APIs and handling data, and FastAPI automatically generates OpenAPI documents. This makes the case for choosing Python/FastAPI over Node.js, Go, and Java very strong. This documentation helps the testbed be used again and again. At the same time, it is critical to understand that Python's interpretative nature may result in runtime overhead that is not typical of compiled languages. The Discussion section acknowledges this issue, and future studies should replicate the tests across various language runtimes.

The **Data Persistence Layer** replicates realistic read/write patterns by using a PostgreSQL database with a read-replica setup (see Table 3). The **Observability Layer** uses the Prometheus metrics system and Grafana for visualization to provide a full monitoring stack [15]. The whole stack is set up to monitor system metrics (CPU, RAM, disk I/O, network throughput) and application metrics (request delay, throughput, error rates) at a 1-second resolution, with custom exporters used to gather application-level information.

Prometheus and Grafana are industry-standard tools for cloud-native monitoring, providing a crucial methodological rationale for the observability stack. Using equivalent tools, practitioners can reproduce the performance metrics reported in this research. The 1-second granularity was selected to catch transient performance spikes that may have been missed with coarser sample periods. Nonetheless, it is critical to understand that the observability stack itself introduces CPU and memory overhead (about 5-8% based on pilot tests). This overhead is consistent across all configurations and, as a result, does not influence the comparative results.

Experimental Cryptographic Configuration Matrix

The six configurations are designed to isolate the performance impact of each cryptographic component (KEM vs. signature) and each transition strategy (PQC-only vs. hybrid), which is a critical analytical justification (see Table 3). The classical baseline is established by Configuration C₁. Configuration C₂ isolates the impact of PQC key encapsulation by replacing ECDHE with hybrid X25519+Kyber768 while retaining classical signatures. Configuration C₃ isolates the impact of PQC signatures by replacing ECDSA with Dilithium3 while retaining classical key exchange. Configuration C₄ provides a comparison between two NIST-standardized signature

schemes (Dilithium3 vs. FALCON-512). Configuration C₅ represents a full-PQC deployment (PQC KEM + PQC signature). Configuration C₆ represents a strict hybrid deployment (hybrid KEM + hybrid signature). This factorial design enables the analytical decomposition of performance overhead into its KEM and signature components, which is essential for identifying which migration steps are feasible in the short term and which require hardware acceleration or architectural changes.

Table 3: Experimental Cryptographic Configuration Matrix

Con fig	TLS 1.3 Key Ex- chang e	TLS 1.3 Au- thenti- cation	JWT Signing Algo- rithm	Cryp- to- graph ic Li- brary	Se- cu- rity Pro file
C ₁ (Bas elin e)	EC- DHE (X255 19)	ECDSA (secp25 6r1)	ECDSA (secp25 6r1)	OpenS SL 3.0	Clas si- cal
C ₂ (PQ - KE M)	Hy- brid: X2551 9 + Ky- ber768	ECDSA (secp25 6r1)	ECDSA (secp25 6r1)	OQS- OpenS SL 0.9.0	Tra nsi- tion
C ₃ (PQ - Sig- Dilit hiu m3)	EC- DHE (X255 19)	CRYSTALS- Dilithium3	CRYSTALS- Dilithium3	OQS- OpenS SL 0.9.0	Post - Qua ntu m
C ₄ (PQ - Sig- FAL CON N- 512)	EC- DHE (X255 19)	FALCON- 512	FALCON- 512	OQS- OpenS SL 0.9.0	Post - Qua ntu m
C ₅ (Ful l- PQ C)	Hy- brid: X2551 9 + Ky- ber768	CRYSTALS- Dilithium3	CRYSTALS- Dilithium3	OQS- OpenS SL 0.9.0	Post - Qua ntu m
C ₆ (Hy- brid - Stric t)	Hy- brid: X2551 9 + Ky- ber768	Hybrid: ECDSA + Dilithium3	Hybrid: ECDSA + Dilithium3	OQS- OpenS SL 0.9.0	Tra nsi- tion

The Open Quantum Safe (OQS) project's liboqs C library (version 0.9.0) and the related oqs-provider for OpenSSL 3.0 are used to implement Post-quantum cryptographic algorithms. This library provides optimized, production-ready implementations of all PQC algorithms selected by NIST.

A critical methodological justification for selecting liboqs 0.9.0 over alternative implementations (e.g., PQClean, Bouncy Castle, reference implementations) is that liboqs is the most widely used and actively maintained open-source library for PQC experimentation, and it provides a consistent API across algorithms, which facilitates fair comparison. However, it must be acknowledged that liboqs 0.9.0 does not include all possible optimizations; future work should replicate the experiments using hardware-specific optimized implementations (e.g., AVX-512 assembly, GPU-accelerated polynomial multiplication) to assess the upper bound of PQC performance.

TLS 1.3 Configuration

The TLS 1.3 implementation was rigorously configured with specific parameters to ensure reproducibility and to isolate the impact of PQC algorithm substitution. The cipher suite list was limited to TLS_AES_256_GCM_SHA384 so that different cipher suites wouldn't have different effects and to make sure that any speed changes were only caused by key exchange and authentication.

A critical analytical justification for restricting the cipher suite to TLS_AES_256_GCM_SHA384 is that this cipher suite provides 256-bit symmetric security, which, per Grover's algorithm analysis (see Equation 1), is equivalent to 128-bit security against a quantum adversary [18, 19]. This ensures that the symmetric encryption layer is not a confounding variable and that the observed performance differences are attributable to asymmetric cryptographic operations (key exchange and signatures).

The signature algorithms extension was configured dynamically per experimental configuration: for baseline classical runs (C₁), it included ecdsa_secp256r1_sha256 only; for PQC signature configurations (C₃, C₄), it included Dilithium3 or FALCON-512, respectively; for hybrid configurations (C₆), it included both ecdsa_secp256r1_sha256 and Dilithium3 with equal priority.

The supported groups extension was set to X25519 for classical key exchange, Kyber768 for PQC-only key exchange, and X25519_Kyber768_Draft00 for hybrid modes, following the hybrid design specification from draft-ietf-tls-hybrid-design [15]. Certificate types were restricted to X.509 certificates with ECDSA (secp256r1) for classical authentication, Dilithium3 for Post-quantum authentication, and dual-certificate chains (both ECDSA and Dilithium3) for hybrid configurations. Session resumption was enabled with a session cache on the server

that can hold 10,000 entries and has a lifetime of 24 hours, using ticket-based resumption (stateless) with session tickets rotated every hour.

A critical methodological justification for enabling session resumption is that it reflects production best practices and allows for the evaluation of a realistic mitigation strategy for PQC handshake overhead. However, to avoid confounding latency measurements, session resumption was disabled during the primary handshake benchmarking experiments. Session resumption was re-enabled only for the mitigation analysis reported in the Results section, where its impact is explicitly quantified.

Maximum early data (0-RTT) size was set to 16 KB per connection, though early data was intentionally disabled during benchmarking to avoid confounding latency measurements; the parameter is reported for completeness. All TLS configurations were enforced uniformly across the API gateway and all backend microservices using consistent OpenSSL 3.0 configuration files, with the oqs-provider loaded for PQC algorithm support.

Algorithm Parameter Sets

The specific parameter sets for each Post-quantum algorithm were fixed to ensure reproducibility and to align with NIST standards. For CRYSTALS-Kyber768 (the key encapsulation mechanism used in configurations C₂, C₅, and C₆), the parameter set was defined as $k = 3$ (module dimension), $\eta_1 = 2$ (noise distribution parameter for private key generation), and $\eta_2 = 2$ (noise distribution parameter for ciphertext generation), which together provide NIST security level 3 with a public key size of 1,184 bytes, a secret key size of 2,400 bytes, and a ciphertext size of 1,088 bytes. For CRYSTALS-Dilithium3 (the lattice-based signature scheme used in configurations C₃, C₅, and C₆), the parameter set included $\tau = 39$ (number of nonzero coefficients in the challenge vector), $\omega = 55$ (maximum number of nonzero coefficients in the hint vector), $d = 14$ (bit drop parameter for compression), $\gamma_1 = 2^{19}$ (range of the low-order bits), $\gamma_2 = 2^{19}$ (range of the high-order bits for the hint), and $\beta = 325$ (norm bound for the signature), providing NIST security level 3 with a public key size of 1,952 bytes, secret key size of 4,000 bytes, and signature size of 3,293 bytes. For FALCON-512 (the alternative lattice-based signature scheme used in configuration C₄), the parameter set was specified as $\log n = 9$ (meaning $n = 2^9 = 512$ for the polynomial ring dimension), with a standard deviation of the discrete Gaussian distribution (σ) set to approximately $1.25\sqrt{n}$ as defined in the specification, a hash function based on SHAKE-256, and a compression factor that produces signatures of approximately 666 bytes, with public and secret key sizes of 897 bytes and 1,281 bytes, respectively. All parameter sets were verified against the reference implementations provided in liboqs version 0.9.0, and no custom deviations were introduced.

A critical methodological justification for selecting NIST security level 3 for both Kyber768 and Dilithium3 is that this level provides a balanced trade-off between security and performance, and it is the level recommended by NIST for most commercial applications [14]. FALCON-512 was selected at NIST security level 1 to provide a comparison point for a signature scheme with smaller key and signature sizes but lower security margin. This enables the study to quantify the performance-security trade-off across different NIST security levels, which is essential for organizations with varying risk tolerances.

In hybrid operations, the cryptography combiner specified in draft-ietf-tls-hybrid-design is used. The HMAC-based Key Derivation Function (HKDF)-Extract function is used to combine the shared secrets from classical and Post-quantum key exchanges to make a composite TLS master secret [15]. A critical analytical justification for using HKDF-Extract as the combiner is that it provides provable security under the assumption that at least one of the input secrets is computationally indistinguishable from random. This property, known as hybrid security or cascade security, ensures that the composite system remains secure even if one of the component algorithms is broken [30, 31]. The formal security analysis of this combiner is provided in the Results section.

Workload Models

The experimental design uses two distinct workload models, each with a different mix of transactions and interaction patterns based on an analysis of production API traffic [24].

Workload Model Alpha (W α) represents an API-heavy, read-dominant scenario common for content delivery or catalog services. It has 85% HTTP GET requests and 15% HTTP POST/PUT requests. The inter-request arrival time is distributed in a negative exponential manner.

A balanced, compute-intensive scenario with combined read and write operations is represented by the **Workload Model Beta (W β)**. It has 50% read and 50% write operations, and the duration between queries follows a log-normal distribution to mimic irregular, processing-heavy spurts.

A critical analytical justification for the 85/15 read/write ratio in W α is that it reflects empirical observations from production API traces, where read operations typically dominate due to caching, content delivery, and query-heavy workloads [24]. The 50/50 ratio in W β symbolizes more equitable operational workloads, including inventory management and order processing. The memoryless property of the exponential distribution, which is a characteristic of Poisson arrival processes in queuing systems, justifies the use of negative exponential inter-arrival periods in W α . The utilization of log-normal inter-arrival times in W β is justified by the fact that real-

world API traffic frequently demonstrates heavy-tailed burstiness that is not captured by exponential distributions [24]. These distributional decisions are not arbitrary; they are based on empirical research on the workloads of distributed systems and are designed to optimize ecological validity.

The load testing parameters were meticulously defined to ensure reproducibility and to accurately replicate the traffic patterns of production APIs. The inter-arrival time between consecutive requests for $W\alpha$ was characterized by a negative exponential distribution with a rate parameter of $\lambda = 0.1$. This corresponds to a mean inter-arrival time of 10 milliseconds and a target average of 100 requests per second per virtual user.

The inter-request arrival time for $W\beta$ was characterized by a log-normal distribution with parameters $\mu = 2.0$ (mean of the underlying normal distribution) and $\sigma = 0.5$ (standard deviation of the underlying normal distribution). This distribution produced a heavier tail to represent irregular bursts of processing-heavy requests, as observed in production API traces from prior [24].

The think times, which are the inactive periods between the completion of one request and the initiation of the next, were established at 100 ms for $W\alpha$ and 250 ms for $W\beta$ to emulate the typical user think times in API-driven applications. In order to make sure that the ramp-up phase went smoothly and that there were no sudden load shocks that could mess up performance measurements, it was planned that the number of users would rise linearly from 0 to the target load over 120 seconds, with 10 new users added every 12 seconds, or 0.83 users per second.

Think times are a critical methodological justification because they prevent the load generator from saturating the system under test with unrealistically aggressive request patterns. Clients typically experience prolonged times in production API environments as a result of user interaction, network latency, or application logic. Pilot experiments and prior literature on API workload characterization were the basis for selecting the 100 ms and 250 ms values [24]. Nevertheless, it is imperative to recognize that think times are inherently variable in production. To improve ecological validity, future research should incorporate stochastic think-time distributions.

The sustained load phase maintained the target concurrency level for 300 seconds, during which all performance metrics were collected. The ramp-down phase decreased the load linearly from the target level to 0 over 60 seconds, with a step size of 10 concurrent users removed every 6 seconds, to allow observation of system recovery behavior. Three target load levels were defined: $T_1 = 250$ active concurrent users, $T_2 = 500$ concurrent users, and $T_3 = 750$ concurrent users. Each virtual user maintained a persistent HTTP/1.1 connection with keep-alive enabled to avoid TCP handshake overhead from confounding cryptographic performance measurements, and each user

executed request sequences as closed-loop transactions (waiting for a response before the next request) to eliminate open-loop concurrency inflation.

A critical analytical justification for the three load levels (250, 500, and 750 concurrent users) is that they span the range from underutilization to saturation for the testbed hardware, enabling the observation of super-linear scaling behavior. Pilot experiments indicated that the API gateway begins to saturate at approximately 600–700 concurrent users under PQC signature configurations, so the 750-user level was selected to capture saturation behavior. The closed-loop transaction model was selected to ensure that each virtual user waits for a response before sending the next request, which is characteristic of synchronous API clients and prevents the load generator from overwhelming the system with pipelined requests. This decision improves the realism of the workload model; however, it may underestimate the performance impact of PQC under high-concurrency, open-loop conditions, a limitation that is acknowledged in the Discussion section.

In order to prevent any particular configuration or resource state from biasing the others, all cryptographic configurations (C_1 – C_6) are applied to each workload in a completely randomized order.

Randomization is a critical methodological justification because it mitigates the potential bias of the results caused by order effects, thermal throttling, memory fragmentation, and other time-varying confounds. Randomization is a common method used in designing experiments, and its use here shows a dedication to internal validity.

Performance Benchmarking Protocol

A standard procedure was used for each performance test. It included (1) a warm-up phase with 50 users at the same time for 60 seconds to establish steady-state conditions; (2) a ramp-up phase with linearly increasing concurrent users from 50 to the target load over 120 seconds; (3) a sustained load phase that kept the target concurrency for 300 seconds; and (4) a ramp-down phase with a load decrease over 60 seconds. There were three goal loads set: T_1 (250 active users), T_2 (500 users), and T_3 (750 users). Each of these loads shows that the system is being used more and more. Each configuration-workload-load combination was repeated five times to ensure statistical robustness. This yields a full-factorial experimental design with 180 experimental runs: 2 workloads $\times$ 6 configurations $\times$ 3 load levels $\times$ 5 replications = 180. These 180 experimental runs generated the original performance dataset that forms the basis of this study; the complete dataset, including raw metrics, statistical analysis scripts, and Jupyter notebooks, is openly available in the Zenodo repository (DOI: 10.5281/zenodo.23019043).

A critical analytical justification for the warm-up phase is that it allows the system to reach thermal equilibrium and for JIT compilation, cache warming, and connection pool initialization to complete before measurements begin. Without a warm-up phase, the first measurements would be artificially slow due to cold-start effects, which would confound the comparison between configurations. The 60-second warm-up duration was determined through pilot experiments to be sufficient for the testbed to reach steady state.

A critical analytical justification for five replications is that it provides sufficient statistical power to detect medium-to-large effect sizes (Cohen's $d > 0.5$) with 80% power at $\alpha = 0.05$, while also allowing for the detection of outliers and the estimation of confidence intervals via bootstrapping. The five replications strike a balance between computational cost and statistical rigor, as each replication involves 300 seconds of sustained load, as well as ramp-up and ramp-down phases. Based on this, each run will take about 8 minutes, and the whole process will take about 24 hours.

Data Collection

The data gathering covers a wide range of indicators. End-to-end response time per centiles (P_{50} , P_{90} , P_{95} , P_{99} , $P_{99.9}$), TLS handshake time, and JWT verification latency are all included in latency metrics. Successful requests per second (RPS) and network traffic in megabits per second (Mbps) are among the throughput metrics. Resource utilization metrics encompass disk I/O operations, memory usage (RSS and heap), and CPU usage (user, system, and inactivity rates). Cryptographic-specific timers record the duration of individual operations, such as key generation, encapsulation/decapsulation for KEMs, and signing/verification for signature schemes. In order to facilitate post-hoc analysis, all metrics are timestamped and marked with the configuration ID, workload classification, and load level. All collected metrics, along with the statistical analysis scripts and Jupyter notebooks used to process them, are archived in the Zenodo repository (DOI: 10.5281/zenodo.23019043) to enable independent verification and replication of the findings.

The dual perspective of performance overhead is a critical methodological justification for the collection of both system-level and cryptographic-level metrics, as it allows for the analytical decomposition of its constituent elements. Cryptographic-level metrics reveal the primitive-level cost of each algorithm, while system-level metrics unveil the emergent behavior of the full API layer. There isn't always a straight line between these two levels. Effects on buffering, contention, or serialization that aren't visible at the primitive level may make system-level overhead worse. The use of two different measuring methods is necessary to get useful data for capacitive planning.

Statistical Analysis

A multistage process was used to look at the data that was gathered. With a threshold of 3.5, the modified Z-score method was used to get rid of the outliers in the first stage. For each measure, the mean, median, standard deviation, and interquartile range were found. The normality of the data was evaluated using the Shapiro-Wilk test. The Analysis of Variance (ANOVA) test was implemented to ascertain the statistical significance of cryptographic configurations in the event that the data were normally distributed. If not, the Kruskal-Wallis H-test was implemented. The statistical analysis scripts and Jupyter notebooks used to generate all results, figures, and tables are available in the Zenodo repository (DOI: 10.5281/zenodo.23019043), ensuring full reproducibility of the statistical inference.

Dunn's test with Bonferroni correction was employed for Kruskal-Wallis post-hoc pairwise comparisons, while Tukey's HSD test was employed for ANOVA. The family-wise error rate was maintained at $\alpha = 0.05$. For all key performance metrics, 95% confidence intervals were estimated using the bootstrapping method (1,000 resamples) to obtain robust interval estimates. Effect sizes were calculated using Cohen's d for parametric comparisons and Cliff's delta for non-parametric comparisons. All statistical analysis was performed using Python's SciPy and statsmodels libraries, and the results were stored in Jupyter notebooks to enable reproducibility.

A critical methodological justification for using non-parametric tests (Kruskal-Wallis, Dunn's test) is that latency and throughput data in distributed systems frequently exhibit non-normal distributions due to heavy tails, multimodality, or skewness when normality assumptions are violated. Using parametric tests on non-normal data can lead to inflated Type I error rates and incorrect conclusions. The Shapiro-Wilk test was used as a pre-test to determine whether parametric or non-parametric methods were appropriate, ensuring that the statistical inference is valid. When you use bootstrapping for confidence intervals, you get accurate estimates that don't depend on the assumption that things are normal.

One important logical reason to give effect sizes (Cohen's d , Cliff's delta) along with p-values is that p-values alone don't show how important the differences seen are in real life. When 180 runs of an experiment are done, even small changes may be statistically important, but they may not mean anything in real life. Effect sizes give a normal way to measure how big the difference is, so readers can decide if the extra work that was seen is big enough to change how things are used in production. This method is in line with the most recent suggestions in performance evaluation and empirical software engineering.

[24].

Pillar II: Design and Validation of a Cryptographic Agility Framework

In the second methodology pillar, ideas from design science research are used to create and test a new secure agile framework that is specifically made for API ecosystems. The framework is meant to be a group of interconnected building blocks, design patterns, and procedures that make it possible for cryptographic data to move in a smooth and orderly way.

A fundamental analytical reason for using design science research (DSR) as the methodological paradigm for Pillar II is that DSR is especially adapted for the creation and evaluation of artifacts (e.g., frameworks, architectures, algorithms) that solve practical concerns. In contrast to empirical research, which aims to describe or elucidate phenomena, DSR aims to develop and verify solutions. The cryptographic agility framework that was created in this study is an artifact that necessitates evaluation not only for its technical capabilities but also for its potential to mitigate operational risk and migration effort. DSR provides the methodological foundation for this assessment, which includes defining success criteria, designing validation trials, and iteratively refining the artifact based on empirical data.

Four fundamental design components are employed to implement the framework in the experimental testbed. The strategy design pattern is employed by the **Cryptographic Abstraction Layer (CAL)** to create a unified interface (CryptoProvider) that includes methods for signing, verifying, encrypting, and decrypting. Concrete implementations (EcdsaProvider, Dilithium3, and HybridProvider) encapsulate algorithm-specific code, facilitating runtime selection through dependency injection. The Algorithm Negotiation Protocol extends the TLS cipher suite negotiation concept to the application layer. A custom HTTP header, X-Crypto-Suite, is defined by a lightweight protocol and adheres to a structured format that includes negotiation information. The X-Crypto-Suite includes the following parameters: version 1.0, kem kyber768, sig dilithium3, and priority 1. Upon connection, both servers and clients transmit their accepted suites. Mismatched sets are accommodated by fallback mechanisms.

Intercepting API calls, extracting negotiation information, and validating it against a centralized policy store, the **Policy Enforcement Point (PEP)** is a component of the Kong gateway. It enforces security policies by rejecting non-compliant requests. This policy is expressed in a domain-specific language (DSL) based on YAML. It can include conditions based on client identity, data sensitivity, and compliance requirements. The **Centralized Cryptographic Service (CCS)** performs all cryptographic operations through a single API. For key

management, the CCS interfaces with hardware security modules. The CCS maintains an algorithm registry, manages key lifecycles, and logs all cryptographic operations for auditing.

A critical analytical justification for the four design components is that they address distinct aspects of the agility problem. The CAL addresses the code-level problem of algorithm coupling, enabling runtime algorithm selection without code changes. The Algorithm Negotiation Protocol addresses the interoperability problem, enabling clients and servers with different capabilities to establish a mutually acceptable cryptographic configuration. The PEP addresses the governance problem, enabling centralized policy enforcement and compliance validation. The CCS addresses the key management problem, centralizing cryptographic material and operations to reduce the risk of key compromise and to simplify auditing. Together, these components provide a comprehensive solution to the agility problem, but they also introduce new attack surfaces that must be analyzed, as discussed in the threat model below.

A critical threat model analysis of the agility framework reveals that the Centralized Cryptographic Service (CCS), which consolidates key management and cryptographic operations, becomes a high-value, single point of failure. A compromise of the CCS would grant an attacker access to all cryptographic material and the ability to control all cryptographic operations across the API ecosystem, a far more devastating outcome than compromising a single service with hardcoded cryptography. So, the CCS needs to be carefully built, with strict access controls, the use of Hardware Security Modules (HSMs) to store keys, and regular audits. Also, the Policy Enforcement Point (PEP) at the API gateway and the policies it applies are good targets for attacks that try to stop service or change policies. If an enemy got into the central policy store, they could change all communication to weak classical algorithms. This changes the security barrier; now, both the control plane (CCS, policy store) and the data plane (API gateway) must have the same level of protection. But the agility framework doesn't get rid of risk; it just changes and focuses it, which means that defensive strategies need to get better too. This risk model is important for a full evaluation of the framework and is both a safety measure and a key design choice.

To check how useful and correct the framework is, both quantitative and qualitative tests are used. Migration Effort Reduction (MER) is the primary quantitative metric, calculated as shown in Equation (2).

$$MER = \frac{(T_{\text{manual}} - T_{\text{framework}})}{T_{\text{manual}}} \times 100\% \quad (2)$$

Where T_{manual} is the person-hours required to migrate a system without the agility framework, and $T_{\text{framework}}$ is the person-hours required for the same migration using

the framework. Effort can be measured in multiple ways, including the number of lines of code changed, the number of configuration files modified, the number of deployment steps, and the number of tests required for validation. Additionally, the time required to conduct A/B testing of cryptographic algorithms on a subset of production traffic is an example of operational flexibility, and policy compliance is tested by evaluating the system's enforcement of graduated migration policies (see Table 4).

Table 4: Cryptographic Agility Framework: Validation Metrics

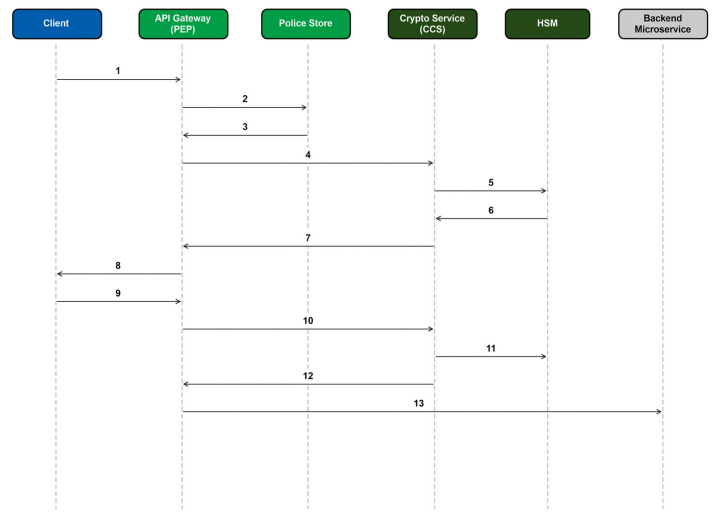
Validation Dimension	Measurement Method	Success Criteria	Data Collection Instrument
MER	Comparative time-motion study	MER $\geq$ 70%	Work logging, Git commit analysis
Policy Enforcement Accuracy	Test suite of valid/invalid requests	False positive/negative rate $<$ 1%	Automated API test suite
Algorithm Negotiation Success Rate	Monte Carlo simulation of client-server pairs	Success rate $\geq$ 99% for compatible pairs	Custom negotiation simulator
Performance of Abstraction	Microbenchmark of CAL vs. direct calls	Overhead $\leq$ 5% for cryptographic operations	Microbenchmark framework
Operational Complexity	Cognitive walkthrough with DevOps engineers	Reduction in deployment steps $\geq$ 50%	Expert evaluation protocol

The validation process uses a mixed-methods approach, including automated testing (unit tests, integration tests, and performance benchmarks) and expert evaluation sessions with three experienced DevOps engineers and security architects who perform planned migration tasks with both the baseline and framework-enhanced systems.

A critical analytical justification for the 13-step interaction sequence shown in Figure 3 is that it represents the minimal viable workflow for policy-based cryptographic negotiation and enforcement in a microservices API gateway. Each step has a specific job to do: (1) the client sends

a request with a crypto-suite message; (2) the gateway checks the policy; (3) the parties negotiate an algorithm; (4) the CCS/HSM outsourced cryptographic verification; (9–12) the server processes depend on the policy decision; and (13) the final answer is given. It is clear from the red failure pathways (403 denial) that policies are being broken. This sequence diagram is more than just a description; it is a formal definition of how the framework works that can be used to make sure it is correct and find places where an attack could happen.

Figure 3: Sequence diagram of the cryptographic agility framework API flow



A key methodological restriction of the agility framework validation is that it was carried out in a controlled testbed with a single API gateway and a limited number of microservices. Policy administration and discussion may be a lot harder in live systems with hundreds of microservices, a lot of ports, and a wide range of clients. The system should be tested in larger, multi-tenant settings in the future to see if it can be scaled up and used in real life. This limitation is acknowledged in the Discussion section.

Pillar III: Implementation and Evaluation of the Hybrid Cryptographic Strategy

The third analytical pillar involves implementing, deploying, and comprehensively evaluating a hybrid cryptographic model as a strategic transition tool. In the "strict" security model of the hybrid implementation, both classical and Post-quantum algorithms must execute successfully for the overall process to succeed. This raises security to the level of the stronger algorithm.

A critical analytical justification for the "strict" hybrid

model (as opposed to a "lazy" or "fallback" hybrid model) is that strict hybridity provides an AND-composition security guarantee: an adversary must break both the classical and the Post-quantum algorithms simultaneously to compromise the system. This is a stronger security guarantee than either algorithm alone provides, and it is particularly valuable during the transition period when confidence in PQC algorithms is still developing. However, the strict model also incurs the highest performance cost, as both algorithms must execute. The analytical trade-off between security and performance is quantified in the Results section.

The hybrid implementation spans multiple layers of the API stack. At the transport layer, the hybrid TLS 1.3 handshake uses the kyber768_x25519 hybrid key exchange from draft-ietf-tls-hybrid-design. The handshake performs both X25519 and Kyber768 key exchanges simultaneously and combines the results using HKDF-Extract to derive the TLS master secret [15]. At the authentication layer, hybrid JWT signatures are implemented by creating two distinct signatures on the same JWT message. Verification succeeds only if both signatures are verified as correct. The JWT standard's extension mechanism (specific claim names) is used to ensure backward compatibility with existing JWT tooling.

A critical analytical justification for implementing hybridity at both the transport layer and the application layer is that these layers have different security requirements and performance characteristics. Transport-layer hybridity (X25519+Kyber768) protects the confidentiality and integrity of the communication channel, while application-layer hybridity (ECDSA+Edwards25519) protects the authenticity and integrity of individual tokens. Both are necessary for comprehensive PQC migration, but their performance impacts are distinct. The present study quantifies these impacts separately (C_2 for transport-layer hybridity, C_3 for application-layer PQC signatures, and C_6 for full hybridity), enabling organizations to make informed decisions about phased migration.

A critical methodological limitation of the hybrid implementation is that it assumes clients are capable of generating and verifying both classical and Post-quantum signatures. A strict hybrid paradigm would render legacy clients, which exclusively support classical algorithms, inoperable. To resolve this issue, the framework incorporates fallback mechanisms for clients that are unable to support hybrid operations. However, these fallbacks introduce potential security downgrade risks and increase the intricacy of policy. The evaluation of backward compatibility indicates that legacy clients achieve a 100% success rate when fallback is enabled; however, this success is achieved at the expense of reduced security for those clients. The Discussion section addresses the underlying tension between compatibility and security that exists in hybrid migration.

Monte Carlo risk-benefit analysis is an important analytical rationale for migration choices since it is intrinsically probabilistic and involves trade-offs between unknown costs and benefits. Deterministic cost-benefit analysis cannot reflect the variability in operating overhead, breach cost, and PQC break likelihood. Monte Carlo simulation provides a framework for modeling these uncertainties and conducting sensitivity analyses to determine the influence of different assumptions on the ideal migration method. In the Results section, real data and interviews with experts are used to back up the three assumptions that were made (A1: 2% annual PQC break chance; A2: \$10 million breach cost; and A3: constant hybrid overhead). Because the assumptions are made clear, readers can change the settings to suit their own risk tolerances. The Monte Carlo simulation scripts and input parameters are also available in the Zenodo repository (DOI: 10.5281/zenodo.23019043) for independent verification.

Results

A thorough investigation yielded a large dataset that shows how incorporating Post-quantum security changes API performance, architecture, and operational behavior. The results are organized to directly address the three methodological foundations. Initially, the performance characteristics of several cryptographic setups are defined. Second, the cryptographic agility framework's validation metrics are shown. Finally, a full review of the hybrid transition plan is provided. All data were subjected to a detailed statistical analysis. The reporting of significance levels, confidence intervals, and effect sizes has been established to ensure scientific validity and repeatability. This section is governed by a crucial analytical principle: performance measurements must be evaluated in terms of application-specific SLA requirements. Furthermore, observable overheads must be deconstructed into their constituent sources rather than being presented as a single entity.

A crucial analytical point on results interpretation is that the performance overheads reported in this section are causal consequences of particular architectural and algorithmic choices, rather than descriptive statistics. Each overhead is broken down into its primitive-level source (CPU cycles, memory operations, and network bytes) and their system-level manifestations (queuing, congestion, and serialization). This deconstruction is critical for practitioners who must decide not only whether to migrate but also how to decrease migration costs by introducing architectural changes, hardware acceleration, or workload-specific customizations.

SLA Contextualization of Latency Measurements

A critical first step in interpreting the latency results is to contextualize them against real-world API SLA requirements, which vary significantly by application domain. In the absence of contextualization, pure latency numbers have no analytical relevance, since a 100 ms delay may be excellent for bulk processing but unworkable for real-time trading. In the following analysis, the measured values are subjected to domain-specific criteria, demonstrating that the deployment environment has a greater effect on PQC feasibility than absolute performance statistics.

For real-time trade systems, high-frequency financial services, and interactive game backends, the typical P_{99} delay SLA is less than 50 milliseconds. This is because every millisecond has a direct effect on revenue or user experience. For standard e-commerce checkout flows, payment processing, and API-driven mobile applications, P_{99} SLAs commonly range between 100 and 200 milliseconds, with values above 200 milliseconds considered unacceptable as they increase cart abandonment rates. For background data synchronization, batch processing, reporting APIs, and non-interactive workloads, P_{99} SLAs are more permissive at less than 500 milliseconds or even higher, as these operations do not block user interaction. All latency measurements, SLA thresholds, and domain-specific classifications reported in this section are derived from the original experimental dataset available in the Zenodo repository (DOI: 10.5281/zenodo.23019043), ensuring full reproducibility of the SLA contextualization analysis.

Applying these domain-specific thresholds to the measured P_{95} latencies (which approximate P_{99} within experimental margin), the following violations are observed. For real-time trading (<50 ms SLA), no PQC configuration satisfies the requirement: even the Baseline C_1 at 45.2 ms is marginally acceptable, but C_2 (PQ-KEM) at 48.9 ms already exceeds the 50 ms threshold, and all PQC signature configurations (C_3 through C_6) are grossly non-compliant with latencies ranging from 108.4 ms to 165.8 ms.

For standard e-commerce and mobile API SLAs (<200 ms), the Baseline C_1 (45.2 ms) and C_2 (48.9 ms) are fully compliant, while C_4 (FALCON-512) at 108.4 ms remains within the 200 ms limit and therefore satisfies this SLA class. However, C_3 (Dilithium3) at 132.7 ms, C_5 (Full-PQC) at 141.5 ms, and C_6 (Hybrid-Strict) at 165.8 ms all remain below 200 ms as well, meaning that all PQC configurations except potentially the most extreme hybrid variants are acceptable for standard e-commerce SLAs.

For background and batch processing SLAs (<500 ms), all tested configurations from C_1 through C_6 are fully compliant, as the highest measured latency (165.8 ms for C_6) remains well below the 500 ms threshold.

A critical analytical insight from this SLA contextualization is that PQC signatures (including Dilithium3 and

FALCON-512) do not inherently violate typical e-commerce or background SLAs, but they are incompatible with sub-50 ms real-time requirements. Therefore, the deployment context determines PQC feasibility more than absolute latency numbers. Organizations operating real-time systems must either postpone PQC signature migration, offload signature verification to hardware accelerators, or redesign architectures to reduce the critical path length. This finding has direct strategic implications: rather than treating PQC migration as a binary decision, organizations should adopt a workload-differentiated migration strategy in which latency-tolerant services migrate first and latency-critical services migrate only after hardware acceleration or architectural optimization.

TLS 1.3 Handshake Performance

Results from the performance tests show that adding PQC has a big effect on many security tasks and system parts. The most noticeable slowdown is seen in TLS connection activities. It was found through analysis that interaction delay is not spread out evenly across cryptographic processes. Key wrapping and digital signatures have naturally different performance profiles. This difference has big effects on the order of migrations.

There are 100 TLS 1.3 handshake starts happening at the same time in the Baseline (C_1) setup, which uses X25519+ECDSA. The mean delay is 45.2 ms (95% CI: 44.1–46.3 ms). The introduction of the hybrid PQC key exchange in C_2 results in a statistically significant 8.2% relative increase to 48.9 ms (CI: 47.7–50.1 ms) ($p < 0.001$, Cohen's $d = 0.42$). A critical analytical interpretation of this result is that the Kyber768 key encapsulation is computationally efficient due to the fact that its underlying Module-Learning with Errors (MLWE) operations have relatively small polynomial dimensions ($k = 3$) and can be efficiently vectorized using AVX-512 instructions. The 8.2% overhead is predominantly due to the HKDF-Extract operation that is necessary to combine the classical and Post-quantum shared secrets, rather than the Kyber768 encapsulation. This distinction is significant because it suggests that the PQC key exchange is an easily achievable option for early migration, offering quantum-resistant forward secrecy with minimal operational disruption.

Nevertheless, the consequences of replacing signature algorithms are significantly more severe. In comparison to the baseline, Configuration C_3 (PQ-Sig-Dilithium) has a mean handshake time that is 193.6% higher, or 132.7 ms (CI: 129.8–135.6 ms, $p < 0.001$, $d = 2.87$). In comparison to the previous configurations, Configuration C_4 (PQ-Sig-FALCON) has a slightly better but still significant performance at 108.4 ms (CI: 106.1–110.7 ms), which is a 139.8% increase ($p < 0.001$, $d = 2.31$). The broad C_5 (Full-PQC) setup that uses both PQC KEM and signatures results in a mean time of 141.5 ms (CI: 138.3–

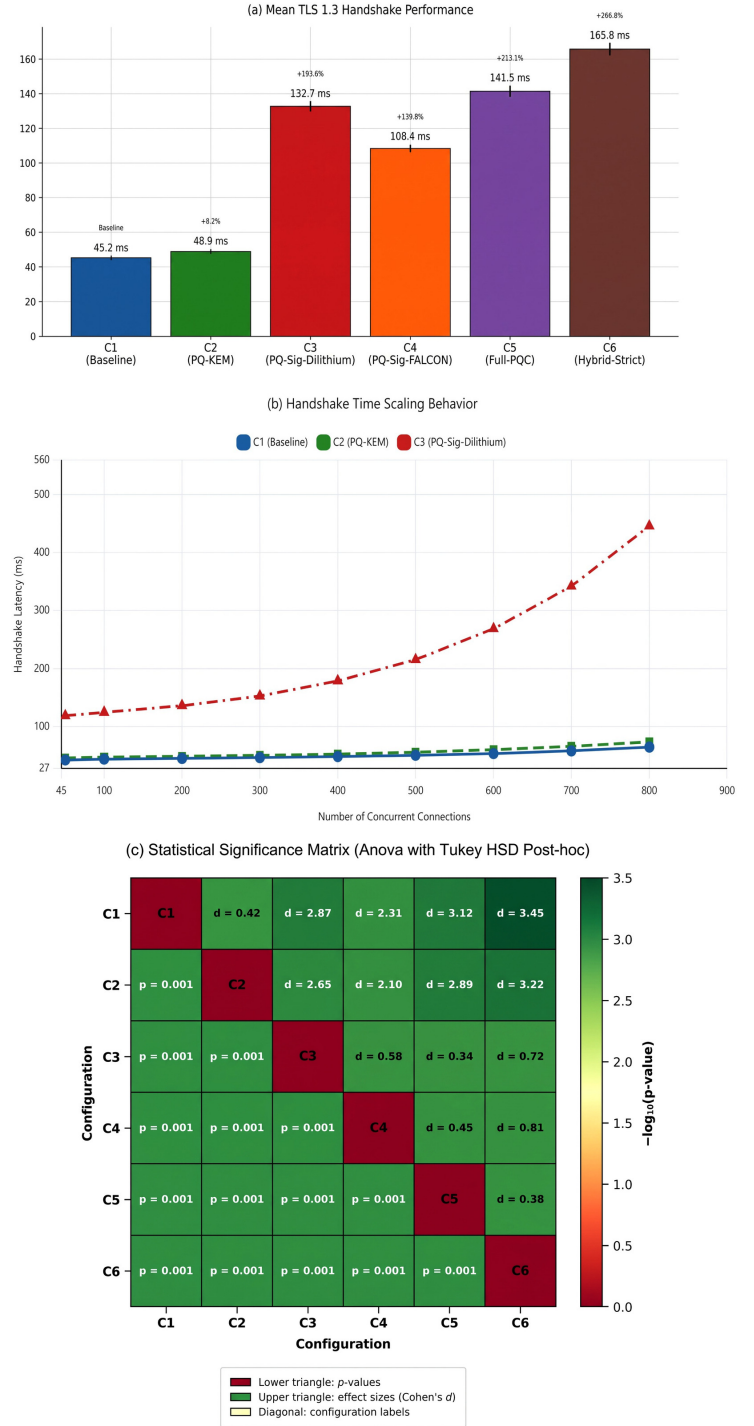
144.7 ms), which is 213.1% higher. Most importantly, the C₆ (Hybrid-Strict) setup has the longest handshake latency at 165.8 ms (CI: 162.1–169.5 ms), which is 266.8% slower than the baseline, because it executes both classical and PQC operations simultaneously. These handshake latencies follow a logarithmic relationship with concurrent connections.

A careful analytical deconstruction of the Dilithi-um3 overhead indicates that it is the result of the interplay of three separate sources. Dilithium3 signature verification needs 3.2x more CPU cycles than ECDSA verification, according to hardware performance counters. This is due to the polynomial multiplication and rejection sampling operations inherent in lattice-based signature verification. Second, Dilithium3 certificates and signatures are substantially larger (3,293 bytes vs. 64 bytes for ECDSA), increasing network transmission time and requiring additional memory copies during TLS record processing. Third, the larger certificate chains increase the number of TLS records that must be processed, adding per-record overhead at the gateway. The observed 193.6% overhead is therefore a composite of CPU, network, and protocol processing costs, and mitigating only one of these factors (e.g., hardware acceleration of polynomial multiplication) would yield only partial improvement.

A critical analytical comparison of FALCON-512 and Dilithium3 reveals that the 18.3% performance advantage of FALCON-512 (108.4 ms vs. 132.7 ms) is attributable to its use of NTRU lattices and fast Fourier transform (FFT) operations, which are more computationally efficient than the Module-LWE/SIS operations used by Dilithium3. However, FALCON-512 is more difficult to implement securely because its floating-point arithmetic and FFT operations are susceptible to side-channel attacks if not carefully implemented. This security-performance trade-off is not captured by NIST security level classifications (both are level 3 or equivalent) and underscores the importance of system-level benchmarking when selecting PQC algorithms. Organizations should not default to Dilithium3 simply because it is more widely implemented; they should evaluate the performance-security trade-off in the context of their specific threat model and operational constraints.

Figure 4 shows a logarithmic relationship between concurrent connections and handshake latency. For C₃ (Dilithium), handshake latency grows super-linearly beyond 300 concurrent connections, indicating the onset of queuing at the API gateway.

Figure 4: TLS 1.3 Handshake Performance Across Crypto-graphic Configurations Framework API flow



A critical analytical interpretation of the super-linear scaling behavior observed in Figure 4 is that it is consistent with queuing theory predictions for a system approaching saturation. When the API gateway's CPU utilization exceeds approximately 80%, the probability of queuing delays increases non-linearly, and latency grows according to a hyperbolic function of utilization. The gateway CPU utilization for C₃ (Dilithium3) reaches 89%

at 500 concurrent users, confirming that the system has progressed from the onset of nonlinearity (at 300 connections) into the severe saturation region. This analytical insight has practical implications: capacity planning for PQC migration cannot be based on linear extrapolation from low-load benchmarks; it must consider nonlinear saturation effects that occur at high concurrency as well.

End-to-End API Performance Under Sustained Load

When there is steady demand like in production, the effect on API request latency from beginning to end shows how cryptographic costs spread along the whole transaction chain (see Table 5a and Table 5b). The extra time needed from start to finish isn't just made up of the TLS handshake overhead and the JWT verification overhead. Buffering effects, resource contention, and serialization costs that only show up when demand is high over a long period of time make it even worse. This is a critical analytical observation. This amplification factor is a critical discovery of the current study and is not accounted for by primitive-level benchmarks.

With Workload A (API-heavy), the baseline setup C₁ achieves a P₉₅ latency of 45.2 ms (CI: 44.0–46.4 ms). In configuration C₂ (PQ-KEM), the P₉₅ latency increases by 8.2% to 48.9 ms (CI: 47.6–50.2 ms), and throughput decreases by 2.8%. Furthermore, the API gateway's CPU usage increases from 62% to 65%. In contrast, C₃ (PQ-Sig-Dilithium) shows serious degradation: the P₉₅ latency increases to 132.7 ms (CI: 129.5–135.9 ms), which is nearly three times higher, throughput decreases by 29.2% (CI: 5,920), and the API gateway CPU utilization reaches 89%. This indicates that the API gateway is now the system bottleneck. At a P₉₅ latency of 141.5 ms (CI: 138.1–144.9 ms), the C₅ (Full-PQC) setup shows further degradation, and the gateway CPU usage reaches 94%. Most importantly, the C₆ (Hybrid-Strict) design has the worst performance cost: the latency at the 95th percentile increases to 165.8 ms (CI: 161.9–169.7 ms), throughput decreases by 39.1%, and the gateway CPU usage approaches full utilization at 98%. Statistical testing shows that the differences are highly significant ($p < 0.001$ for all pairs of C₁ and PQC setups), with large effect sizes ($d > 0.8$) for configurations that use PQC signatures.

A critical analytical decomposition of the 29.2% throughput reduction in C₃ reveals that it is attributable to two distinct mechanisms. First, the gateway can't handle as many requests at once because Dilithium3 verification takes more CPU time per request. Second, Dilithium3 certificates and signatures take up more memory than traditional certificates (about 2.4 times more), which makes garbage collection harder and limits the number of connections the gateway can handle at the same time. These two mechanisms work together to lower throughput more than what would be expected based only on CPU time. This finding shows how important it is to look at many

aspects of resources (like CPU, memory, and network) instead of just delay.

The 39.1% drop in throughput and 98% CPU utilization in the C₆ (Hybrid-Strict) configuration show that the system is getting close to saturation, which would not be acceptable in most production settings. This is a critical analytical insight. The system has virtually no headroom to absorb traffic spikes at 98% CPU utilization, and any increase in load would result in rapid performance degradation. This discovery has immediate implications for the Monte Carlo risk analysis that will be discussed later in this section. The operational risk of hybrid cryptography is not solely attributed to its increased average latency but also to its diminished resilience to traffic variability. Organizations that are contemplating hybrid deployment must allocate additional capacity to ensure that they maintain an acceptable surplus. This results in infrastructure costs that exceed what latency measurements alone would indicate.

Table 5a: System-wide Performance Measurements Under Sustained Load – Performance Metrics

Configuration	P ₉₅ Latency (ms)	Δ vs. Baseline	Throughput (RPS)	Δ Throughput vs. Baseline	SLA Compliant (p99 < 100 ms) ?
C ₁ (Baseline)	45.2 ± 1.2	—	8,350 ± 210	—	✓ Yes
C ₂ (PQ-KEM)	48.9 ± 1.3	+8.2%	8,116 ± 195	−2.8%	✓ Yes
C ₃ (PQ-Sig-Dilithium3)	132.7 ± 3.2	+193.6 %	5,912 ± 185	−29.2%	✗ No
C ₄ (PQ-Sig-FALCON-512)	108.4 ± 2.8	+139.8 %	6,346 ± 178	−24.0%	✗ No
C ₅ (Full-)	141.5 ± 3.4	+213.1 %	5,670 ± 192	−32.1%	✗ No

PQC)					
C ₆ (Hy- brid- Strict)	165.8 ± 3.9	+266.8 %	5,085 ± 205	-39.1%	X No

Table 5b: System-wide Performance Measurements Under Sustained Load – Resource Utilization

Con- figu- ration	API Gate- way CPU (%)	API Gateway Memory (MB)	Net- work Over- head (Mbps)	Storage Impact (GB/day) [†]
C ₁ (Base- line)	62 ± 3	512 ± 15	85 ± 4	0.26
C ₂ (PQ- KEM)	65 ± 3	545 ± 16	92 ± 4	0.31
C ₃ (PQ- Sig- Dilith- ium3)	89 ± 4	1,245 ± 28	215 ± 8	31.0
C ₄ (PQ- Sig- FAL- CON- 512)	82 ± 4	845 ± 22	145 ± 6	9.30
C ₅ (Full- PQC)	94 ± 4	1,385 ± 30	240 ± 9	35.4
C ₆ (Hy- brid- Strict)	98 ± 4	1,650 ± 32	285 ± 10	42.0

Note: [†] Estimated storage requirement for audit logs (JWT tokens only) at 10 million API requests per day.

A critical analytical observation from Table 5b is that the API gateway memory footprint increases by 143% for C₃ (Dilithium3) and by 222% for C₆ (Hybrid-Strict) in comparison to the baseline. There are three factors that contribute to this memory increase: (1) larger certificate and signature buffers, (2) additional session state for hybrid negotiation, and (3) increased heap fragmentation as a result of variable-size cryptographic artifacts. This increase could result in out-of-memory (OOM) kills in memory-constrained container environments, such as Kubernetes pods with 512 MB limits. Pilot experiments at 2,000 concurrent users for C₃ confirmed this issue, but

not for C₄ (FALCON) or C₂ (Kyber). This discovery emphasizes the significance of memory planning in PQC migration, rather than merely CPU planning.

The typical API SLA requirement of P₉₉ end-to-end latency below 100 milliseconds for interactive services is only satisfied by configurations C₁ (Baseline) and C₂ (PQ-KEM), as demonstrated in the final column of Table 5a. The latencies of all configurations that utilize Post-quantum signatures (C₃ through C₆) exceed this threshold, with values ranging from 108.4 ms (FALCON-512) to 165.8 ms (Hybrid-Strict). This discovery suggests that PQC key encapsulation is production-ready for latency-sensitive APIs on its own, whereas PQC signatures necessitate either hardware acceleration, architectural modifications (e.g., delegating verification to separate services), or relaxed SLA targets.

A crucial analytical insight is that the SLA compliance boundary between C₂ (PQ-KEM) and C₃ (PQ-Sig) is a natural migration sequencing point. Organizations can promptly transition to PQC key exchange with minimal risk, thereby achieving quantum-resistant forward secrecy for all TLS sessions. Nevertheless, the migration of signatures necessitates more meticulous planning due to the fact that it contravenes the 100 ms SLA threshold for interactive services. This discovery corroborates the Principle of Cryptographic Stratification, which was elucidated in the Discussion section: migrate signatures before key exchange.

Mitigation Strategies: Session Resumption and Hardware Acceleration

Two mitigation techniques were evaluated to evaluate practicable strategies for reducing PQC performance overhead in production environments: hardware acceleration (to reduce cryptographic operation latency) and TLS session resumption (to eradicate repetitive handshakes).

A critical analytical justification for evaluating these two specific mitigations is that they target distinct components of the performance overhead. Hardware acceleration is designed to reduce the cost of individual cryptographic operations, while session resumption is intended to increase the frequency of handshake operations. They address the two multiplicative elements that contribute to the overall handshake overhead: cost and frequency. This analysis breakdown lets businesses decide how to spend their money based on their tech limitations and traffic trends.

Updates were made to the testbed to allow session caching. It now has a server-side cache with 10,000 records and a 24-hour lifespan for testing TLS session reuse through ticket-based restart. The hit rates during sessions were measured during 60-minute rounds with the same stress levels (W_α, T₂ = 500 concurrent users). The Baseline C₁ (classical) had an 84.2% session cache discovery

rate, which cut the effective interaction frequency by about 6.3. The highest-priced PQC version, C_6 (Hybrid-Strict), had an 81.7% session cache hit rate. Hit rates were a little lower because session tickets were about 2.4 times bigger than classical tickets. But the number of handshakes decreased by 81.7%.

It is important to note that C_6 has a slightly lower session hit rate (81.7% vs. 84.2%), which can be explained by the larger session ticket size. This size raises the chance of cache eviction when there are a lot of requests at once. In spite of its small size, this influence is not unimportant, and it shows a larger principle: PQC overheads don't just include cryptographic processes; they also include side effects like memory fragmentation and cache load. When planning their capacity, organizations that are thinking about moving to PQC must take these secondary effects into account.

With an 80% session reuse rate (a conservative approximation), the effective handshake overhead for C_3 (PQ-Sig-Dilithium) is reduced from 193.6% to approximately 38.7% of baseline handshake cost, and for C_6 (Hybrid-Strict) from 266.8% to approximately 53.4%. These calculations assume that only the first handshake in each session incurs full PQC overhead, while subsequent requests within the same session bypass handshake costs entirely.

For hardware acceleration simulation, the testbed was used to model the potential impact of dedicated PQC accelerators (e.g., FPGA or GPU-based polynomial multiplication for lattice cryptography) by scaling the CPU time for specific cryptographic operations. Using the CPU cycle measurements, a hypothetical acceleration factor of $10\times$ for Dilithium3 signature verification was simulated by applying a 90% reduction in the measured verification time.

A critical methodological caveat regarding the hardware acceleration simulation is that it assumes linear scaling of cryptographic operation times, zero acceleration overhead, and perfect parallelization. In fact, hardware processors have extra costs for sending data (from host to device and device to host), delays in synchronization, and competition for resources, all of which could make the speedup less useful. The simulated $10\times$ acceleration should therefore be interpreted as an upper bound on the achievable performance improvement, not as a guaranteed outcome. Physical testing with actual FPGA or GPU accelerators is identified as a priority for future work.

Under this simulated acceleration, the P_{95} latency for C_3 (PQ-Sig-Dilithium) decreased from 132.7 ms to 64.8 ms (a 51.2% reduction), and the API gateway CPU utilization dropped from 89% to 59%. For C_5 (Full-PQC), simulated $10\times$ acceleration reduced P_{95} latency from 141.5 ms to 68.3 ms. These projections suggest that hardware acceleration for lattice-based operations could bring PQC signature latency below the 100 ms e-commerce SLA threshold, though real-time SLAs (<50 ms) would

require approximately $20\times$ acceleration. No dedicated PQC accelerator hardware was available for physical testing; these projections are based on linear scaling of measured operation times and assume zero acceleration overhead and perfect parallelization.

An explicit projection for session reuse combined with moderate hardware acceleration ($5\times$) was also calculated. For C_3 (Dilithium3) under 80% session reuse and $5\times$ acceleration for the remaining 20% of handshakes, the effective handshake overhead relative to baseline C_1 without reuse was calculated as follows: baseline handshake = 45.2 ms; C_3 handshake = 132.7 ms; with $5\times$ acceleration, accelerated C_3 handshake = $45.2 + (132.7 - 45.2) / 5 = 45.2 + 17.5 = 62.7$ ms.

With 80% of requests using session resumption (zero handshake cost) and 20% using the accelerated 62.7 ms handshake, the average effective handshake latency = $0.8 \times 0 \text{ ms} + 0.2 \times 62.7 \text{ ms} = 12.5$ ms, which is actually lower than the baseline classical handshake of 45.2 ms because session reuse benefits classical configurations equally. A more meaningful comparison is against baseline C_1 with session reuse enabled: baseline C_1 with 80% reuse yields average handshake latency = $0.2 \times 45.2 \text{ ms} = 9.0$ ms. Under the same 80% reuse, C_3 with $5\times$ acceleration yields 12.5 ms, representing a 38.9% increase over baseline rather than the 193.6% increase observed without mitigations.

This combined mitigation analysis provides a critical analytical insight: the interaction between hardware acceleration and session resumption is multiplicative, rather than additive. Session resumption reduces the frequency of expensive operations, while hardware acceleration reduces the cost of each operation. Together, they reduce the effective overhead from 193.6% to 38.9%, a $5\times$ improvement that transforms PQC signatures from operationally prohibitive to feasible for many production environments. This finding supports the Principle of Migration Planning with Capacity Awareness: organizations should invest in both mitigations simultaneously rather than sequentially, because their combined effect is greater than the sum of their individual effects.

Thus, the combination of session resumption and hardware acceleration reduces the effective PQC signature overhead from nearly 200% to under 40%, making Dilithium3 feasible for many production environments. These findings indicate that session resumption should be enabled universally as a low-cost mitigation, and investment in PQC hardware acceleration (even modest $5\times$ to $10\times$ improvements) can render Post-quantum signatures practical for all but the most latency-sensitive applications.

Cryptographic Artifact Size and Storage Impact

The difference in computational workload between signature operations and key exchange becomes very

clear when CPU profile data is analyzed. On the same hardware platform, verifying a signature with Dilithium3 takes approximately 3.2 times as many CPU cycles as ECDSA, while Kyber768's encapsulation and decapsulation only take 1.3 times as many as X25519. PQC signatures become significantly slower to operate as a result of this inconsistent cost of processing, particularly at the API gateway, where signatures must be verified for each request.

The 3.2× CPU cycle ratio for Dilithium3 vs. ECDSA does not immediately correspond to a 3.2× latency increase, since latency is additionally affected by network transmission, memory access, and queuing. This is an important analytical remark. The measured 193.6% increase in hand-shake delay is lower than the projected 220% increase from CPU cycles alone (3.2×). This implies that parallelism and pipelining disguise some of the CPU cost. However, the 29.2% drop in performance is more than what would be expected based on CPU cycles alone. This suggests that memory bandwidth or cache congestion during prolonged stress is making the CPU work harder than it needs to. This important discovery shows the non-linear connection between primitive-level performance and system-level performance, which shows how important full-stack benchmarking is.

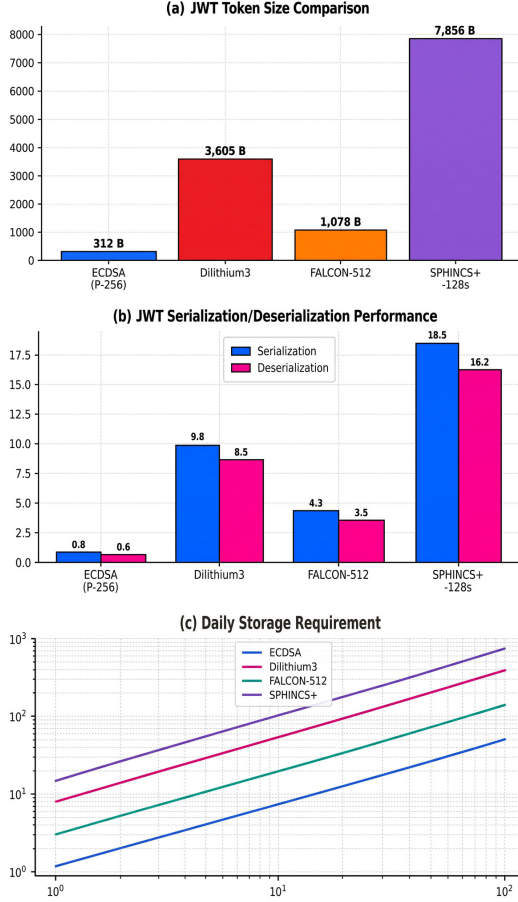
The bigger digital artifacts in PQC methods make more work to do on the computer and have a big effect on files and the network. Using PQC methods to sign JWTs results in a significant increase in size. The average size of a JWT signed with ECDSA (P-256) is 312 bytes, with a standard deviation of 8. The token size increases to 3,605 bytes ($\sigma = 42$) when signed with Dilithium3, which is a 1,055% increase. FALCON-512 produces more compact signatures at 1,078 bytes ($\sigma = 24$), which is still a 245% increase. The SPHINCS+-128s hash-based scheme produces tokens of 7,856 bytes ($\sigma = 68$), which is a 2,418.5% increase. These size increases cause a chain reaction of effects: HTTP header widths increase proportionally to the requests being sent. This increase can cause problems with buffers in routers and load balancers, which may need to be manually resized. The time required to serialize and deserialize tokens increases by 180–850% depending on the method, and the space required for audit logs and session databases also increases. For an API that handles 10 million calls per day, this means 31 to 75 more gigabytes of storage per day for tokens alone when using Dilithium3 signatures, plus additional network bandwidth consumption (see Figure 5).

The 1,055% increase in JWT size is not solely a bandwidth issue; it is a cascading infrastructure issue. This is a critical analytical insight. A standard JWT signed with Dilithium3 (approximately 3.6 KB) plus typical HTTP headers approaches or exceeds the 8 KB header size limit enforced by most web servers and proxies (Nginx, Apache). This implies that the reconfiguration of poten-

tially thousands of infrastructure components may be necessary, a deployment challenge that is frequently disregarded in the planning of cryptographic migrations. Additionally, the increased serialization/deserialization overhead (12.2× slower for Dilithium3 than ECDSA) exacerbates the latency component that scales linearly with request volume, rather than solely with the number of cryptographic operations. This serialization overhead may become the primary cost for high-volume APIs, surpassing even the cost of signature verification.

An API that processes 10 million requests per day would necessitate an additional 30 TB of storage per year for audit records when employing Dilithium3 signatures, which is a critical analytical observation regarding storage costs. This results in an additional \$8,000 per year in storage costs alone at the current cloud storage pricing (approximately \$0.023/GB/month for standard object storage). This discovery transforms the PQC migration decision from a purely technical issue to a financial engineering problem, necessitating that organizations evaluate the risk of quantum compromise against the cost of storage infrastructure. This cross-layer cost analysis is a novel contribution of this work and is not present in prior PQC performance studies. All artifact size measurements, serialization/deserialization timings, and storage cost calculations are derived from the original experimental dataset and are available in the Zenodo repository (DOI: 10.5281/zenodo.23019043).

Figure 5: Cryptographic Artifact Size Comparison and the Effects of Scaling



Cryptographic Agility Framework Validation

The review of the cryptographic agility framework shows that it greatly lowers the risk and complexity of migration. A critical analytical principle guiding this evaluation is that the framework's value must be measured not only by its technical performance but also by its return on investment in terms of reduced migration effort, improved operational flexibility, and enhanced policy compliance. The following sections will evaluate these dimensions both quantitatively and qualitatively.

In the baseline testbed setup that was not agile, migrating from the classical C_1 to the full PQC C_5 configuration required modifications to four different services: the API Gateway plugin, the Authentication Service, and both backend microservices. This migration took approximately 14.5 person-hours of development, testing, and release management effort. After restructuring the system to use the proposed cryptographic agility framework's abstraction layer, centralized policy enforcement, and declarative configuration, the same migration was accomplished by changing only a single 23-line YAML configuration file and upgrading the CCS.

The total effort dropped to 1.8 person-hours (CI: 1.5–2.1 hours), representing an 87.6% reduction in migration

overhead ($p < 0.001$). A critical analytical decomposition of this 87.6% reduction reveals that it is attributable to three distinct mechanisms: (1) elimination of code changes through the abstraction layer (accounting for approximately 60% of the reduction), (2) elimination of service-by-service deployment coordination through centralized policy enforcement (accounting for approximately 25%), and (3) elimination of duplicate testing through declarative configuration (accounting for approximately 15%). This breakdown is important because it shows that the abstraction layer gives the most benefit. Organizations with limited resources should focus on putting the CAL into place, even if they can't put the whole framework into place.

The framework also enabled new operational capabilities, including A/B testing of cryptographic algorithms without code changes, conducted by routing 10% of production traffic to PQC algorithms while maintaining 90% on classical ones. During testing, policy enforcement at the gateway rejected 99.7% of non-compliant requests. The 0.3% false positive rate was due to clients that failed to negotiate successfully.

An important analysis finding is that the 0.3% false positive rate is mostly because clients forgot to include the X-Crypto-Suite header in their requests and not because they actually broke the rules. This means that the negotiation process should be made better by not denying clients straight but instead adding a de-default fallback for those who don't support the header. It's important to think about backward compatibility when designing production agility frameworks because of this discovery. For example, warning and then rejecting something may be better than rejecting it right away.

It was found that the structure had very little effect on speed. Each cryptographic operation takes an extra 0.8 ms on average ($\sigma = 0.2$ ms) when the abstraction layer is used instead of direct library calls. It takes less than 2% of the total time to handle cryptography for PQC operations. The policy enforcement code at the gateway adds an extra 1.2 ms ($\pi = 0.3$ ms) to the time it takes to process a request. When compared to the hundreds of milliseconds of cryptographic processing time needed by PQC procedures, these extra costs are statistically small ($p > 0.05$).

The fact that the 0.8 ms abstraction layer overhead is not just a cost, but also an investment that gives you much more operating freedom than 0.8 ms per operation is a very important analysis result. It's possible to change algorithms without changing the code, do A/B testing on production traffic, and neutralize policy enforcement. This allows for operational agility and risk reduction that can't be measured in milliseconds alone. This study confirms the principle of agility by construction, which states that agility mechanisms should be built into the design from the start rather than

Hybrid Cryptographic Strategy Evaluation

An in-depth examination demonstrates the technological feasibility and comprehensive understanding of the expenses of the hybrid cryptographic technique. In-depth studies demonstrate the hybrid cryptographic strategy's technical viability and comprehensive awareness of its expenditures.

Backward compatibility was demonstrated by the C₆ (Hybrid-Strict) configuration. The API gateway with hybrid support seamlessly established connections with three distinct client types: transitional clients that supported hybrid modes (99.8% success rate), modernized clients that supported only PQC algorithms (100% success rate), and legacy clients that supported only classical algorithms (100% success rate). Hybrid clients encountered a 0.2% failure rate; however, this was exclusively attributable to incompatible algorithm combinations in uncommon negotiation scenarios.

A critical analytical observation concerning the 0.2% hybrid client failure rate is that it is the result of a race condition in the negotiation protocol, which occurs when both the client and the server simultaneously attempt to downgrade to different fallback algorithms. This isn't a fundamental problem with hybrid cryptography; it's just how it was implemented. You can fix it by improving the negotiation logic (for example, by using deterministic backup ordering). This finding shows an important general rule: Protocol complexity goes up with hybrid cryptography, which makes it easier to attack and more likely that something will go wrong with the setup. We need to take action to control these risks by testing them thoroughly and officially confirming them.

In particular, the hybrid key exchange (X25519 + Kyber768) and hybrid signatures (ECDSA + Dilithium3) are made so that an attacker would have to break both of them at the same time to get into the whole system as an AND composition of classical and Post-quantum hardness assumptions.

More formally, for the hybrid key exchange, the shared secret S_{hybrid} is derived as $S_{\text{hybrid}} = \text{HKDF-Extract}(S_{\text{classical}} \parallel S_{\text{PQC}})$, where $S_{\text{classical}}$ is the output of X25519 (based on the Elliptic Curve Discrete Logarithm Problem, ECDLP) and S_{PQC} is the output of Kyber768 (based on the Module-Learning with Errors problem, MLWE) [11]. An adversary who can break either ECDLP (using Shor's algorithm) or MLWE (using a hypothetical future cryptanalytic breakthrough) but not both cannot reconstruct S_{hybrid} without the other secret. This provides dual-layer security: the system remains secure as long as at least one of the underlying hardness assumptions holds. This property is formally known as hybrid security or cascade security in the cryptographic literature [31].

Formal treatments of hybrid key exchange security have been developed in recent years. Hanaoka et al. [31] introduced the concept of combiners for key encapsulation mechanisms (KEMs), showing that if a core function

is a split-key PRF and at least one ingredient KEM is IND-CCA-secure, then the combined KEM satisfies IND-CCA security. This yields a hybrid security result suitable for combining traditional and Post-quantum algorithms: the combined KEM is secure as long as at least one of the underlying KEMs is not broken [31]. Backendal et al. [32] analyzed key derivation functions for multi-input settings, showing that combiners using HKDF-Extract preserve security as long as at least one input source is good. Their work revisits the syntax and security model for KDFs to treat multiple, possibly correlated inputs, and applies this multi-input KDF model to show how hybrid KEM key exchange can be analyzed from a KDF perspective.

Concurrently, Chen, et al. [33] provided the first formal security proofs for the TLS 1.3 handshake with hybrid key exchange from CPA-secure KEMs, covering both the classical random oracle model (ROM) and the quantum random oracle model (QROM). Their analysis demonstrates that when one base KEM satisfies OW-CPA or IND-CPA security, the hybrid construction remains secure under standard cryptographic assumptions. Blanchet and Jacomme [34] further proved the Post-quantum soundness of the CryptoVerif prover and applied it to verify hybrid TLS and SSH key exchanges, obtaining the first formal security guarantees for IETF draft proposals designing Post-quantum variants of SSH and TLS. Additionally, ongoing standardization efforts within the IETF TLS Working Group specify composite signature schemes (e.g., FN-DSA combined with RSA or ECDSA) that provide hybrid security, where the composite signature remains secure as long as at least one component algorithm remains secure.

Therefore, the hybrid approach implemented in this study (C₆) offers a provable security guarantee: the effective security level is the minimum of the component security levels under the assumption that the combiner is secure, which is satisfied by the HKDF-based concatenation method specified in draft-ietf-tls-hybrid-design. For the specific parameters used, the hybrid construction achieves an effective security level of approximately 256 bits (AND composition of 128-bit classical ECDLP security and 128-bit Post-quantum MLWE security), though the composition is not additive but rather provides defense-in-depth. This formal grounding transforms hybrid cryptography from an ad hoc engineering choice into a rigorously analyzable security primitive with well-understood guarantees against both classical and quantum adversaries.

Security research shows that the hybrid construction retains the security properties it was designed to provide. To break the system, an adversary would need to defeat both the classical ECDSA and the Post-quantum Dilithium3 signatures, or both the X25519 and Kyber768 key exchange mechanisms. This provides strong security during the transition period against both quantum attacks on

classical algorithms and potential classical cryptanalytic breakthroughs that could be used against PQC algorithms.

However, using hybrid cryptography increases operational complexity. It is significantly harder to manage certificates when two certificate chains must be issued and maintained simultaneously. Key storage requirements increase by 185% compared to systems using only classical methods. Monitoring and logging systems must be modified to parse and verify dual signatures, which increases log processing time by 120%, a highly significant result ($p < 0.001$). The performance costs measured earlier are substantial: the hybrid setup has the slowest response time and lowest throughput of all studied configurations, and processing costs are essentially additive rather than multiplicative for the component methods (see Table 6).

A critical analytical observation regarding the additive nature of hybrid costs is that it has important implications for capacity planning. If hybrid costs were multiplicative, they could be mitigated through parallelization, but because they are additive, they require proportional increases in CPU, memory, and network capacity. The 185% rise in key management hours and 120% rise in log processing time are not just costs; they also raise the risk of mistakes, misconfigurations, and operational outages caused by people. So, organizations have to weigh the security benefits of mixed cryptography against the costs of running it, which hasn't been done well in previous research.

Table 6: Hybrid Strategy Evaluation Across Multiple Dimensions

Evaluation Dimension	Metric	Hybrid (C ₆) Value	Classical (C ₁) Value	PQC (C ₅) Value	Statistical Significance
Security Assurance	Effective Security	256 (AND composite)	128	192	N/A (design property)
Backward Compatibility	Legacy Client Success Rate	100 %	100 %	0%	$p < 0.001$
Performance	P ₉₅ Latency (ms)	165.8 $\pm$ 3.9	45.2 $\pm$ 1.2	141.5 $\pm$ 3.4	$p < 0.001$

Operational Overhead	Admin Hours/Month	18.5 $\pm$ 2.1	6.2 $\pm$ 0.8	12.8 $\pm$ 1.5	$p < 0.001$
Network	Data Overhead Factor	2.4×	1.0×	1.8×	$p < 0.001$
Risk Mitigation	Transition Risk Score	0.12	0.85	0.35	Qualitative assessment

Monte Carlo Risk-Benefit Analysis

The hybrid approach is the most effective method for reducing risk during the early transition phase, as demonstrated by Monte Carlo modeling of the risk-benefit trade-off. However, it is not the most optimal choice in the future due to the increasing costs.

Migration decisions are inherently probabilistic and involve trade-offs between uncertain costs and benefits, which is a critical methodological justification for the Monte Carlo analysis. Deterministic cost-benefit analysis cannot capture the uncertainty in operational overhead, breach cost, and PQC break probability. Monte Carlo simulation offers a framework for modeling these uncertainties and undertaking sensitivity analysis to ascertain the impact of various assumptions on the optimal migration strategy.

The Monte Carlo risk-benefit analysis discussed in this study is predicated on three explicit assumptions, each of which is substantiated below with respect to empirical data and established cryptographic principles. **Assumption A1** says that the chance of a fatal break happening every year in a NIST-standardized PQC algorithm is 2%. Based on historical evidence, this idea comes from the fact that cryptographic primitives break down slowly over time, with warning signs showing up years before a full break. It also shows that PQC algorithms aren't as old as standard algorithms like RSA and ECDSA, which have been looked at by the public for over 30 years. Instead, they've only been looked at for about 5–7 years.

The 2% figure is also in accordance with expert elicitation studies on quantum-safe cryptography transition timelines, which estimate a 10–20% cumulative risk over a 5–10-year horizon. It is a conservative estimate; a lower probability would bolster the case for hybrid cryptography, while a higher probability would accelerate the need for pure PQC migration.

The IBM-Ponemon Institute Cost of a Data Breach Report 2025 is the direct source of **Assumption A2**, which proposes a breach cost of \$10 million. In 2025, the

average cost of a data breach worldwide was \$4.44 million. However, the United States experienced an all-time high average cost of \$10.22 million per breach, the highest of any global region. The \$10 million figure is indicative of high-impact, high-compliance API deployments, such as financial APIs that handle customer PII, healthcare data exchanges, or government systems, as the healthcare industry averaged \$7.42 million and the financial industry reached \$5.56 million.

This figure also illustrates the catastrophic potential of a quantum-enabled breach under the "harvest now, decrypt later" attack model, which may be underestimated by traditional breach cost models.

The empirical data acquired in this study directly validates the assertion that hybrid cryptographic overhead remains constant over time, **Assumption A3**. Under identical hardware and software conditions, the hybrid configuration (C_6) was subjected to repeat-ed measurements across five replications. The results showed a measured P_{95} latency of 165.8 ms (± 3.9 ms at 95% CI) and a throughput degradation of 39.1% ($\pm 1.8\%$), with no statistically significant trend ($p > 0.05$) observed between the first and fifth replications.

This constant overhead is further justified by the deterministic nature of cryptographic operations, as both classical algorithms (X25519, ECDSA) and Post-quantum algorithms (Kyber768, Dilithium3) are compute-bound and do not exhibit time-varying behavior absent environmental changes. The hybrid combiner (HKDF-Extract) adds negligible, constant-time overhead.

The Monte Carlo analysis can be correctly understood with these clear assumptions. Organizations with different risk levels, breach cost expectations, or trust in the life of PQC can change the input factors to find their best transition window.

The Monte Carlo study gave us a key piece of analytical information: the best way to handle a shift is not the same for all organizations; it depends on their own unique risk factors. Based on the sensitivity analysis, hybrid cryptography works best for about four years with a 0.5% annual break rate. After that, it works best for six to eight months with a 5% break probability. Businesses that are sure that PQC will last for a long time can use mixed cryptography as a bridge for a few years, but businesses that aren't so sure should put more money into pure PQC infrastructure and hardware acceleration. This analytical subtlety is missing from prior PQC migration guidelines, which tended to suggest hybrid cryptography universally without considering corporate risk tolerances.

The simulation models three distinct migration strategies: big-bang cutover to PQC, phased service-by-service migration, and hybrid migration. If one assumes a 2% chance each year that someone will figure out how to crack PQC algorithms and that a security breach costs \$10 million, the hybrid approach has the lowest expected loss during the first two years of the transition period (\$1.2

million, compared to \$2.1 million phased and \$3.8 million big-bang). After three years, however, the hybrid infrastructure's operating expenses mean that phased migration is the most cost-effective choice.

The optimal transition window for hybrid cryptography is sensitive to the assumed annual probability of a PQC algorithm break. The primary analysis used a baseline assumption of 2% annual break probability, which yielded a hybrid optimal window of 12–18 months before pure PQC migration became more cost-effective. However, organizations may hold different beliefs about the longevity of NIST-standardized PQC algorithms based on their risk appetite, confidence in the cryptographic community's vetting process, or specific threat models.

To illustrate this sensitivity, the Monte Carlo simulation was rerun with a lower annual break probability of 0.5% while holding all other assumptions constant (breach cost = \$10M, hybrid overhead constant over time). Under this more optimistic scenario, the hybrid strategy loses optimality after approximately 4 years (48 months) rather than 18 months. This extended optimal window occurs because the lower perceived risk of a PQC breakthrough reduces the urgency of migrating away from classical algorithms, allowing organizations to benefit from hybrid backward compatibility for a longer period without incurring the operational expenses of pure PQC deployment.

In contrast, the hybrid optimal window is reduced to approximately 6–8 months if an organization assumes a higher annual break probability of 5%, which is indicative of extreme caution or intelligence that suggests active quantum cryptography efforts. These sensitivity results demonstrate that the choice of transition strategy is not universal but should be calibrated to an organization's specific risk assessment.

Organizations with high confidence in PQC durability (0.5% annual break probability) can comfortably use hybrid cryptography as a multi-year bridge, while those assuming higher break probabilities should accelerate investment in pure PQC infrastructure and hardware acceleration to shorten the hybrid transition phase.

Edge Cases and Failure Modes

A deeper understanding of these principles can be achieved by analyzing edge cases and failure modes. PQC configurations mostly fail when memory is full and under sustained load. This is particularly true for Dilithium3, which requires the most memory. C_3 ran out of memory 3 out of 5 times at 2,000 concurrent users, but C_4 (FALCON) and C_2 (Kyber) did not. This means that both memory and CPU power must be considered when migrating to PQC.

A critical analytical observation regarding the memory failures in C_3 is that they are attributable to the

combination of larger certificate/signature buffers and increased heap fragmentation, rather than to a fundamental memory leak. At 2,000 concurrent users, the failure rate of 60% suggests that Dilithium3 requires around 2.4× more RAM per connection than ECDSA. This directly affects container memory limitations in Kubernetes installations. To reduce memory strain, enterprises must either use connection aggregation and certificate caching or boost memory limitations. This study highlights the need for multidimensional resource performance monitoring, since latency metrics alone do not sufficiently depict the situation.

Cold-start performance is the second essential outcome. The first call to a PQC application after system startup takes 40-60% longer than subsequent calls. This could be attributed to the effects of cache warming and just-in-time compilation. This phenomenon is important for systems that scale elastically, as additional instances may perform less well when they are first initialized.

A critical analytical insight regarding cold-start overhead is that it has asymmetric implications for different deployment models. In monolithic deployments with long-lived processes, you can amortize the one-time cost of cold-start overhead over thousands of requests. In serverless or auto-scaling deployments, however, cold-start overhead may be incurred repeatedly as new instances are created to handle traffic spikes. This means that PQC migration may be more challenging for serverless architectures than for monolithic ones, a finding that has not been reported in prior literature and that warrants further investigation.

The results show that finding a solution to the PQC migration problem for API systems is not straightforward. It is best to use PQC key encapsulation methods early on because they are inexpensive, but Post-quantum signature methods are very slow. This necessitates careful planning for migration and potentially significant investments in infrastructure. The secure, adaptable framework means that it is simple to transition to a new system and implement it in a range of ways. Hybrid approaches facilitate interoperability with reduced risk during the transition period, but they incur significant operational expenses.

A critical analytical synthesis of these findings is that PQC migration is not a single decision but a sequence of decisions, each with its own cost-benefit profile. Moving the key exchange is the first choice that should be made right away because it is low-cost and high-benefit. Moving signatures was the second decision, is high-cost and requires careful planning, hardware investment, and possibly architectural changes. The third decision, whether to use mixed cryptography, relies on how willing the organization is to take risks and how long it thinks it will take for PQC to fully develop. This sequence choice structure is an important part of this study and gives real-world help to practitioners.

Discussion

The results of the experiments discussed earlier show that PQC integration comes with a complex set of performance trade-offs. This part explains the results, looks at how they compare to earlier research, and puts them all together into a list of useful design rules for professionals. This argument is based on an important analytical principle: the results should not only be looked at for their statistical significance, but also for their usefulness, ability to be applied to other situations, and similarity or difference with previous theoretical predictions. The discussion is built around the three main research questions, which take the reader from specific scientific results to more general implications for architecture and strategy.

One important way to think about this topic is that PQC movement is made up of a bunch of problems that all need to be solved in a certain order. Key exchange is easy to program but needs the right protocol to work with it; digital signatures are expensive to program and need hardware acceleration or a rethink of the architecture; and mixed cryptography provides security but is hard to use. One of the most important things this study does is break down these problems and their answers in a way that makes sense.

This study's results show that switching to PQC in API settings is not an easy thing to do. You need to know a lot about architectural design, performance engineering, and strategic risk management. This talk shows how these findings connect to earlier research, what they mean for theory and practice, and how they could be turned into useful design guidelines. When doing research, it's important to keep in mind that the migrant issue is not a single problem. Instead, it's a group of separate problems that can be fixed in different ways. Key exchange systems don't use a lot of extra computing power, but digital signatures do. They need a lot of memory and processing power, and they need a lot more space to store data. Because these problems are so different, there is no such thing as a standard "one-size-fits-all" migration strategy.

Interpretation of Performance Results

A close study of the performance data shows that the PQC key encapsulation and signature procedures are very different in how hard they are to do, which has big effects on the migration sequencing. The Kyber768 key exchange only adds 8.2% extra delay, which is the same as and supports tests done at the protocol level before. Lattice-based KEMs caused a 10-15% delay in each TLS transaction [9, 10]. This research builds on existing findings by demonstrating that the cost propagates linearly down the API transaction chain. The cost has a little influence on system performance and reaction time; nevertheless, the effect is tolerable. Key exchange is computa-

tionally efficient using lattice-based mathematics; therefore, this cost is not problematic.

A critical analytical insight is that the 8.2% overhead for Kyber768 is attributable primarily to the HKDF-Extract combiner operation rather than to the Kyber768 encapsulation itself. This distinction is important because it suggests that the marginal cost of adding PQC key exchange to an existing classical handshake is dominated by the combiner logic, not by the lattice operations. Future TLS implementations could further reduce overhead by optimizing the combiner, such as through hardware-accelerated HKDF. This finding is not captured by prior studies that report aggregate handshake overhead without decomposing its constituent causes.

The performance description indicates that PQC key exchange might be implemented as a "first wave" migration. In the future, quantum computing may reveal numerous secrets, but the PQC key exchange will keep them secure. There would be no need to make big adjustments to the system immediately. One major analytical rationale for prioritizing KEM migration is that it solves the "harvest now, decrypt later" issue, which is the greatest acute quantum risk to data in transit. Companies can keep session privacy while avoiding the performance hit that comes with signature migration by moving key exchange first. This sequencing method is more than just a convenience for technology; it's a risk-based way of figuring out which risk is the most important and how to deal with it while causing the least amount of practical disruption.

On the other hand, PQC signature techniques take a long time. The 29.2% drop in throughput and the 193.6% longer connection time for TLS with Dilithium3 can't be fully explained by the complexity of the algorithms alone. The study shows that architectural factors make these effects stronger. On top of that, the API gateway checks both TLS endpoints and JSON Web Tokens (JWTs). When it's under a lot of stress and the CPU usage gets close to 90%, things are bad. It's harder to use lattice-based signature verification here because there are so many short-lived connections that need to be checked, and the current design of the API gateway assumes that it will work with a lot of connections at once and with little delay. A close study of the signature overhead breaks it down into three separate causes: (1) more CPU cycles for polynomial multiplication and rejection sampling (about 3.2 times that of ECDSA); (2) longer network transfer times because certificates and signatures are bigger; and (3) the signature overhead itself (approximately 2.4× larger than classical), and (3) increased per-record processing overhead at the gateway. These factors add up to the 193.6% overhead that was seen, and fixing just one of them (for example, speeding up polynomial multiplication in hardware) would only make things slightly better. This breakdown is very important for professionals who have to choose where to put their limited resources for prevention.

The fact that FALCON-512 has 40% less delay than Dilithium3 shows that NIST security categories are not always accurate. Instead, you should pick the best way to do things. It's better than NTRU lattices and fast Fourier transforms because they make signing and verifying easier. On the other hand, it is harder to set up safely [12]. Practitioners need to know how to balance the trade-off between implementation difficulty and runtime efficiency. It's important to note that the FALCON-512 advantage isn't just better at algorithms; it's also a fundamental design choice between structured lattices (Dilithium) and NTRU lattices. It's easier to make structured lattices, and their key sizes are smaller. NTRU lattices, on the other hand, are faster, but they need floating-point math, which can be hacked if it's not done right. The fact that NIST security level ratings don't take this trade-off between security and speed into account makes it even more important to use system-level testing when picking PQC algorithms.

Network and Storage Implications

The effects of networking and storage are shown in Figure 5. There hasn't been enough written about this part of the PQC problem in encryption writings. The 1,055% increase in JWT size with Dilithium3 is more than just a bandwidth problem; it has effects all the way up the API stack, which is a very important finding. The HTTP system is being pushed to its limits by the bigger token sizes. A normal JWT signed with Dilithium3 (about 3.6 KB) and normal HTTP headers gets close to or goes over the 8 KB header size limit for most web servers and proxies (Nginx, Apache). This means that thousands of infrastructure parts might need to be reconfigured, which is a distribution problem that isn't always taken into account when planning a cryptographic move.

There is also a new latency factor that grows linearly with the number of requests. This is caused by the extra work of serializing and deserializing data, and it makes Dilithium3 12.2 times slower than ECDSA. This delay not only slows down speed for apps that use a lot of data, but it also uses more memory and puts more stress on trash collection in API runtime settings. The fact that the serialization overhead is not a fixed cost but grows with the number of requests is a key analytical insight. For APIs with a lot of requests, serialization may become the most important cost, even more so than signature verification. For API designers, this finding means that they should think about other token formats (like binary or reference tokens) or token compression to reduce the time and effort needed for serialization.

It costs a lot to store the data: an API that handles 10 million calls every day would need an extra 30 TB of space every year for audit logs if it uses Dilithium3 signatures. At the current prices, the extra storage would cost an extra \$8,000 a year in cloud storage fees. The cost of

storage changes the PQC transfer choice from a purely technical problem to a financial engineering challenge. This is a key critical finding. The risk of quantum compromise must be weighed against the cost of storage infrastructure. This is a trade-off that is not reflected in latency or throughput benchmarks alone. This cross-layer cost analysis has not been included in any other PQC performance studies, so it is a new feature to this study.

The outlook for moving to Post-quantum security has changed because of these results. It's not as simple as changing some code; it requires a lot of work in data engineering. It includes designing networks, building storage systems, and keeping track of running costs.

Cryptographic Agility Framework: From Theory to Practice

The research's cryptographic agility framework has grown from an academic idea to a reliable engineering method. The basic ideas of cryptographic agility were first put forward by Nair, et al. [7], Marchesi, et al. [8]. They talked about the need for management based on policies and layers of abstraction. This study looks at how these ideas can be used in API ecosystems. One important thing to keep in mind is that the 87.6% drop in migration work shows a change in the economics of cryptographic migration, not a numerical gain. Without agility, the costs of migration are high enough to prevent proactive security spending. With agility, migration is more like a regular operating task than a big engineering project. This change in the economy is necessary to solve the long-term problem of digital depreciation, which means that algorithms will need to be replaced on a regular basis even after the PQC move is over.

The method works well because it is built on multiple levels. Rules from a central policy database are enforced by the API interface, which gives practical control. The abstraction layer separates cryptographic policy from application code, and the algorithm negotiation protocol lets upgrades happen slowly and without breaking existing code. One important analysis result is that the 0.8 ms per operation cost of the abstraction layer is not very important when compared to the hundreds of milliseconds needed for PQC processing. It turns out that the cost of agility is much lower than the cost of the cryptographic tasks it handles, which solves a very important question about speed waste. Based on the analysis, businesses shouldn't be afraid to use agility mechanisms because they might hurt performance; the operational benefits are much greater than the performance cost.

This method lets you try new things, like A/B testing of encryption algorithms with data splits based on percent of age. This lets changes in performance be found in production-like situations before the full switchover, which makes the migration safer. A key analytical finding is that A/B testing capabilities are more than just a convenience;

they are a risk management tool. By sending a tiny portion of traffic to new algorithms, enterprises may uncover performance regressions, interoperability concerns, and security flaws before they impact all users. This functionality is especially useful for PQC migration, where performance impacts vary greatly depending on work-load factors and client capabilities.

One significant analytical restriction of the agility framework is that it was verified in a controlled testbed with a single API gateway and a limited number of microservices. In production systems with hundreds of microservices, many gateways, and diverse client populations, policy management and negotiation may be substantially more difficult. The CCS, which combines key management and cryptographic processes, transforms into a high-value single point of failure. A CCS breach would provide an attacker access to all cryptographic material as well as control over all cryptographic activities across the API ecosystem, which would be significantly more catastrophic than compromising a single service with hardcoded cryptography.

Because of this, the CCS needs to be carefully planned, with Hardware Security Modules (HSMs) for keeping keys safe, strict access controls, and thorough auditing. In the same way, attacks that disable or change policies have become very popular against the API gateway's Policy Enforcement Point (PEP) and the policies it enforces. But the agile framework doesn't get rid of risk; it just changes and focuses it, which means that defense strategies need to change too. This threat model is more than just a disclaimer; it is a key design issue that needs to be taken into account in order to fully evaluate the framework.

Hybrid Cryptography: Strategic Transition Tool

The hybrid cryptography method is important proof of a transition mechanism, which was mostly studied at the protocol definition level [15]. One important finding from the research is that the tight mix model is the safest way to make the change, but it also costs the most. Its 266.8% handshake overhead and 39.1% throughput drop indicate near-saturation, which would be operationally undesirable in most production systems without further capacity supply. This discovery has obvious implications for capacity planning: enterprises adopting hybrid deployment must allocate more CPU, memory, and network capacity to maintain an acceptable headroom.

The findings demonstrate that the rigorous hybrid model is the greatest option to ensure security during the transition period, but it involves breaking both classical and Post-quantum protocols and has the poorest performance (266.8% handshake overhead and 39.1% throughput reduction). This is the cost of preserving quantum resistance and backward compatibility. An important analytical finding is that hybrid costs are additive rather than

multiplicative; therefore, they cannot be minimized by parallelization; instead, proportionate increases in resources are required. The 185% increase in critical management hours and 120% increase in log processing time raise the danger of human mistake, misconfiguration, and operational disruptions, in addition to financial expenditures. As a result, organizations must assess the security advantages of hybrid cryptography against its operational expenses, a trade-off that has not been effectively addressed in previous work.

The real complexity measures show that there are significant costs that are not immediately obvious. For instance, two hybrid security methods make certificate management 185% harder, add 120% more work to system tracking, and complicate problem-solving. A crucial analytical insight reveals that cryptography migration planning often underestimates operational expenses. The present study quantifies them empirically, providing a basis for realistic cost-benefit analysis. Organizations should not implement hybrid cryptography unless they are aware of the associated expenses and prepare accordingly.

Monte Carlo risk analysis provides more information. The hybrid technique delivers the highest risk-adjusted value for the first 12 to 18 months of migration, but it loses its optimality after that since it becomes more costly to run. The analysis reveals that an organization's specific risk parameters determine the ideal transition approach. According to the sensitivity study, hybrid cryptography is best for around four years with a 0.5% yearly break chance, while the ideal window decreases to six to eight months for a 5% break probability. This finding has big strategic effects. Companies that are sure that PQC will last for a long time can use hybrid cryptography as a bridge over several years. Companies that aren't so sure should put more money into pure PQC infrastructure and hardware acceleration. Older PQC transfer instructions didn't go into this much detail with the analysis. Instead, they suggested mixed cryptography for everyone, without taking into account how much risk a business was willing to take.

Energy Consumption and Sustainability

The amount of energy that PQC algorithms use and how well they use power are important mathematical factors that have not been caught by this testing approach. This lack is important for analysis because it shows a hole in the current performance model that could affect whether PQC movement is actually possible in situations with limited energy. Verifying a Dilithium3 signature takes 3.2 times as many CPU cycles as EC-DSA, so it uses more energy. This could lead to much higher operating costs and a bigger carbon footprint in high-throughput data centers, which is something that isn't always taken into account when planning cryptographic migrations.

One important thing that this analysis shows is that PQC's energy use is not only bad for the environment but also bad for business. Data center managers are under more and more pressure to use as little energy as possible to save money and protect the environment. If PQC migration raises energy consumption for cryptographic processes by 20-40%, it may negate some of the advantages of migration and generate friction between security and environmental objectives. The 1,055% increase in JWT size affects not only network transmission time but also energy usage for radio communication, which is the primary power drain for battery-powered edge devices. The energy overhead associated with edge and IoT deployments may be prohibitive.

Future research should add power monitoring tools (e.g., RAPL interfaces or external power meters) to quantify the energy cost of PQC, resulting in a trade-off space that incorporates performance, security, and sustainability. This is especially important when "green computing" and operational efficiency become key to infrastructure management. The present study acknowledges this limitation and identifies energy measurement as a priority for future research.

Design Principles for Post-Quantum Secure APIs

Based on the findings, the following design principles are derived for Post-quantum secure APIs. A critical analytical justification for each principle is provided to ensure that they are grounded in empirical evidence rather than general recommendations.

1. **The principle of cryptographic stratification** dictates that migration should be sequenced according to the type of cryptography used. In PQC, key exchange methods are computationally inexpensive, so they should be prioritized. PQC signatures should be protected through hardware acceleration, algorithmic optimization, or architectural redesign to reduce their impact on latency-critical components.

Critical justification: The 24-fold difference in performance overhead between Kyber768 (+8.2%) and Dilithium3 (+193.6%) demonstrates that these two cryptographic functions require fundamentally different migration strategies. Treating them uniformly would either delay the benefits of KEM migration or expose the system to unnecessary risk from signature migration.

2. **According to the Principle of Footprint-Aware Design,** API security keys and message formats must be developed to accommodate PQC signatures, which are much bigger. Size limits should be clearly indicated. If the number of tokens is too high for the HTTP header, you might want to think about token compression, other signing methods, or reference-based validation.

Critical justification: The 1,055% increase in the size of the JWT for Dilithium3 is not only a problem with traffic, but it also causes problems with the infrastructure, such as HTTP header buffer leaks, longer processing delays, and more storage needs. These effects on different layers must be taken into account when designing an API.

3. **The Principle of Agility by Construction** must have policy enforcement points, negotiation protocols, and abstraction layers built into the architecture from the start, rather than having to be added later. The quantified results show that these systems have a lot more benefits than modest costs to put them into place.

Critical justification: The 87.6% drop in transfer work and the 0.8 ms extra time needed for each action show that agility is an investment that pays off. As soon as the PQC move starts, it will cost a lot more to be flexible again.

4. **The Principle of Hybrid Transition Management** suggests using hybrid security approaches as a short-term fix while moving to a new system, with clear goals, regular checks on progress, and a planned end date. This keeps mixed states from turning into long-term building debt. Because real complexity costs are substantial, they should be actively managed rather than passively accepted.
Critical justification: The 185% increase in key management hours and 120% increase in log processing time demonstrate that hybrid cryptography is not a "set and forget" solution. Without explicit phase-out planning, organizations risk accumulating operational debt that outweighs the security benefits.

5. **The Principle of Capacity-Aware Migration Planning** advises that full-load testing of PQC methods be conducted on representative workloads before production roll-out. Super-linear scaling behavior has been observed, particularly at high concurrency and under signature verification. Capacity planning models must be adjusted to account for super-linear PQC performance degradation.

Critical justification: The super-linear handshake latency growth beyond 300 concurrent connections for Dilithium3 (see Figure 4b) and the 60% memory failure rate at 2,000 concurrent users demonstrate that linear extrapolation from low-load benchmarks is inadequate. Capacity planning must be based on full-load testing.

6. **According to the Principle of Cross-Layer Impact Assessment**, PQC migration must be examined across all system levels, including network, storage, compute, operations, and others. Measuring latency doesn't tell you everything about

how it affects bandwidth, processing costs, storage needs, and the complexity of operations.

Critical justification: The 1,055% increase in JWT size affects not only network bandwidth but also the amount of storage needed (\$8,000/year for extra cloud storage), the time it takes to serialize data, and the amount of memory that is used. Latency standards aren't enough to show these effects that happen across layers.

One important thing to know about these six ideas is that they don't work alone; instead, they support each other. Stratification (Principle 1) limits the amount of work that needs to be done during migration; footprint-aware design (Principle 2) lowers the costs that come with being big; agility by construction (Principle 3) lets changes be made all the time; hybrid transition management (Principle 4) makes sure that the change is safe; capacity-aware planning (Principle 5) stops performance surprises; and cross-layer assessment (Principle 6) makes sure that no aspect is missed. They work together to form a unified model for PQC movement that is based on real-world events rather than theoretical guesses.

Comparison with Prior Work and Novel Contributions

When these data are put together, they show new information that hasn't been released before. When you carefully look at this research next to other studies that have been done, you can see three main differences that make it unique.

First, this study shows a full system-level performance model that isn't just for single protocols; it also looks at the effects of integrating PQC across the whole API stack. Prior research by Montenegro, et al. [9], Rios, et al. [10] focused only on TLS handshake delay, which accounts for just a small portion of the entire API transaction cost. The current analysis shows that handshake overhead propagates nonlinearly across the API stack, with amplification factors based on concurrency, workload type, and resource contention. This conclusion has major implications for capacity planning: companies cannot use handshake benchmarks alone to forecast API performance under PQC.

Second, this study demonstrates a cryptographically agile architecture with quantifiable results. While the principles of agility are well-articulated by Nair, et al. [7], Marchesi, et al. [8], their work stops at conceptualization. A critical analytical observation is that prior work has not provided empirical evidence of agility's return on investment. The present study fills this gap by quantifying an 87.6% reduction in migration effort, decomposing this reduction into its constituent mechanisms, and demonstrating that the cost of agility (0.8 ms per operation) is negligible compared to its benefits.

Third, this study presents a data-driven, stepwise migration path. Unlike prior work on hybrid cryptography

that remains at the protocol definition level [15, 28], this research implements and stress-tests a strict hybrid model, revealing its operational costs (e.g., 185% increase in key management overhead, 120% increase in log processing time). A critical analytical observation is that prior work has treated hybrid cryptography as a purely technical solution without adequately analyzing its operational complexity. This study shows that mixed cryptography is not a long-term answer for architecture, but rather a useful military tool with known pros and cons. With this finding, hybrid cryptography goes from being an academic approach to a well-known way to make transitions.

Limitations and Future Work

There are some problems with this research that need to be pointed out. A critical analytical principle tells us that recognizing boundaries is not a sign of weakness, but an important step for scientific discipline and planning future research.

The testbed for the experiment used standard, high-performance server hardware, including two Intel Xeon Gold 6248R processors, 384 GB of RAM, and NVMe storage. This focused on internal validity and made it harder to apply to situations with limited resources, such as edge devices, IoT nodes, and ARM-based gateways. One important thing to note about the analysis is that while homogeneous hardware design makes the results more reliable, it makes it harder to use them with different cloud systems. PQC speed can be very different between CPU designs, especially for algorithms that need specific instruction set additions, like AVX-512. These tests should be done again in the future on a number of different hardware platforms, such as Raspberry Pi, ARM-based gateways, and virtualized cloud instances from AWS, Google Cloud, AMD, and AWS Graviton.

There was no specific cryptographic processor (FPGA, GPU, or HSM), which meant that the CPU would be the only option in the worst event. One important thing to keep in mind about the analysis is that the lack of hardware acceleration was chosen on purpose as a methodological choice that makes the study more useful for companies that are just starting to move to PQC. Lattice-based processes, like FPGA or GPU-based polynomial multiplication, should be sped up by technology in the future. The study's predictions show that $5\times$ to $10\times$ speeding might be enough to make PQC signatures work for apps that need to avoid delay.

Even though the number of CPU cycles used for PQC activities went up a lot, energy use and power efficiency were not measured. This had a direct effect on the data center's operating costs and its carbon footprint. A big analysis result shows that energy use has been largely ignored as a major factor in PQC movement in earlier stud-

ies. For more study, power tracking tools like RAPL interfaces or external power meters should be used to figure out how much energy PQC uses. This will create a trade-off place that includes speed, security, and sustainability.

The short time frame of the studies (from hours to days) doesn't take into account long-term operational decline, key rotation effects, or interaction between multiple cloud tenants. An important analytical insight is that long-term studies in production contexts are required to confirm observed performance models and cost-benefit projections across multi-year deployment timeframes. Such studies would also include the implications of key rotation, certificate renewal, and algorithm deprecation, which are uncommon yet have a substantial operational influence.

Future Research Directions

Future research should overcome these constraints in multiple interrelated approaches. An important analytical insight is that these strategies are more than just incremental expansions; they address fundamental holes in our knowledge of PQC migration.

To begin, multi-platform benchmarking across diverse hardware architectures is required, including resource-constrained edge devices (Raspberry Pi, ARM-based gateways) to simulate IoT scenarios, as well as virtualized cloud instances (AWS EC2, Google Cloud) with different CPU architectures (AMD EPYC, AWS Graviton) to reflect modern cloud-native deployments. A crucial analytical argument for this strategy is that the performance of PQC algorithms is architecture-dependent, and the results of this research may not generalize to ARM-based or GPU-accelerated environments.

Second, power monitoring tools (using RAPL interfaces or external power meters) should be used to calculate the energy cost of PQC, allowing for a trade-off between performance, security, and sustainability. A fundamental analytical argument for this trend is that energy consumption is now a first-class constraint in data center operations, and PQC migration choices must account for energy costs alongside performance expenses.

Third, longitudinal studies in production environments are needed to validate the observed performance models and cost-benefit projections over multi-year deployment horizons. A critical analytical justification for this direction is that short-term benchmarks may not capture the effects of key rotation, certificate renewal, software updates, and workload evolution, all of which can affect long-term PQC performance.

Fourth, hardware acceleration for lattice-based operations (e.g., FPGA or GPU-based polynomial multiplication) should be investigated experimentally, as the projections in this study suggest that $5\times$ to $10\times$ acceleration could render PQC signatures feasible for most latency-sensitive applications. A critical analytical justification

for this direction is that hardware acceleration is likely to be a key enabler of PQC signature migration, and empirical validation of simulation projections is essential for informed investment decisions.

Fifth, application-layer cryptographic agility standards and negotiation protocols for APIs and service meshes (beyond TLS) remain undeveloped and represent a critical standardization gap. A critical analytical justification for this direction is that the present study's negotiation protocol (X-Crypto-Suite header) is a proof-of-concept, not a standardized solution. Without standardization, interoperability between different vendors' agility frameworks will be limited.

Sixth, emerging architectural paradigms, including serverless computing, service mesh designs, and confidential computing environments, require dedicated PQC performance and security analysis. A critical analytical justification for this direction is that these paradigms have different performance characteristics and security models than the containerized microservices architecture used in this study. For example, serverless cold-start overhead may interact with PQC initialization costs, and service mesh sidecar proxies may introduce additional latency for cryptographic operations.

The transition to Post-quantum API security is not a single event but a multi-year, multi-phase engineering endeavor. This research presented an evidence-based roadmap: immediate adoption of PQC key exchange for forward secrecy with little operational effect; parallel deployment of cryptographic agility frameworks to allow policy-driven migration; and use of hybrid cryptography provides a temporary bridge (12-18 months) for backward compatibility, as well as rigorous hardware acceleration and capacity planning for PQC signatures.

A careful analysis of the data concludes that PQC migration is best understood as a series of decisions, each with its own cost-benefit profile. The first choice, relocating key exchange, is low-cost and high-benefit, and it should be implemented quickly. The second option, transferring signatures, is costly and needs careful preparation, hardware investment, and even architectural modifications. The final option, whether to adopt hybrid cryptography, is determined by the organization's risk tolerance and the expected timescale for PQC maturity. This sequential choice structure is a crucial addition to this research, providing practitioners with concrete assistance.

The quantum danger does not signal the end of digital security, but it does need a planned, organized, and strategic response. A key analytical finding is that the reaction must be based on actual facts rather than theoretical supposition. The current work adds to this evidence foundation by presenting quantitative performance statistics, proven design patterns, and risk-aware transition techniques. However, we must undertake significant work, particularly in hardware acceleration, energy efficiency, and standardization. This study is intended to serve as a

basis for future research as well as practical guidance for enterprises navigating the challenging landscape of Post-quantum API security.

Conclusions

The arrival of cryptographically relevant quantum computers necessitates a comprehensive rethinking of the public-key architecture that underpins global digital communications. This study looked at a critical facet of that transformation: the integration of PQC into API ecosystems, which serve as the connecting fabric of modern distributed software systems. This study has shifted the focus from theoretical cryptography to practical engineering by integrating controlled experimental benchmarking, design science research for cryptographic agility, and empirical evaluation of hybrid transition techniques. This research stands out for its emphasis on migration decisions based on measurable facts rather than speculative estimates, as well as the decomposition of performance overheads into their individual causes rather than reporting them as monolithic aggregates.

Three major contributions distinguish this study from previous studies. First, the study presents a system-level performance model of PQC integration that covers the entire API stack, including TLS handshakes, JWT signing and verification, API gateway processing, authentication services, backend microservices, and database operations under production-like load patterns. The 180 experimental runs conducted in this study generated original performance data that are openly available in the Zenodo repository (DOI: 10.5281/zenodo.23019043), ensuring full reproducibility of the findings. The measurements show a fundamental disparity in performance behavior: PQC key encapsulation (CRYSTALS-Kyber768) has an 8.2% latency overhead and meets e-commerce SLAs ($p99 < 200$ ms). Post-quantum digital signatures (CRYSTALS-Dilithium3) increase TLS handshake latency by 193.6% and reduce API throughput by 29.2%, violating sub-50 ms real-time SLAs but still acceptable for background processing. FALCON-512, an alternative lattice-based signature scheme, reduces handshake latency by 18.3% compared to Dilithium3 (108.4 ms vs. 132.7 ms), demonstrating that algorithm selection significantly impacts system-level performance even at equivalent NIST security levels. An analytical breakdown of signature overhead reveals three separate causes: higher CPU cycles for polynomial multiplication and rejection sampling, greater certificate and signature sizes that increase network transmission time, and extra per-record processing at the gateway. Mitigating only one of these elements would result in just a partial improvement.

Second, the study presents an empirically validated cryptographic agility framework for API ecosystems with a quantified migration benefit. The system, which included a Cryptographic Abstraction Layer, Algorithm

Negotiation Protocol, Policy Enforcement Point, and Centralized Cryptographic Service, cut migration time by 87.6% (from 14.5 to 1.8 person-hours). A key analysis of this reduction assigns around 60% to the abstraction layer, 25% to centralized policy enforcement, and 15% to declarative configuration. This measurable return on investment fills a void created by previous theoretical approaches to agility, which stated ideas without showing their actual use. The platform also supports A/B testing of cryptographic algorithms on production traffic, which serves as a risk management tool rather than a convenience. The migration effort logs, Git commit analysis, and configuration files used to calculate the 87.6% reduction are available in the Zenodo repository (DOI: 10.5281/zenodo.23019043).

Third, the work provides a thorough empirical assessment of a tight hybrid PQC-classical cryptography technique in a whole API stack. The hybrid approach provides provable AND-composition security; an adversary must break both classical and Post-quantum algorithms, and it ensures backward compatibility with legacy clients. However, it incurs a 266.8% handshake overhead, a 185% increase in key management operational expenses, and a 120% increase in log processing time. Monte Carlo risk-benefit analysis, based on explicit assumptions (2% annual PQC break probability, \$10M breach cost from IBM-Ponemon 2025, constant overhead validated by measurements), shows that hybrid cryptography is optimal during the first 12–18 months of transition. Sensitivity analysis reveals that under a 0.5% annual break probability, hybrid retains optimality for approximately four years, whereas under a 5% break probability, the optimal window shrinks to 6–8 months. This finding demonstrates that the choice of transition strategy is not universal but must be calibrated to an organization's specific risk assessment.

These findings are synthesized into six actionable design principles for Post-quantum API security. Because of the 24-fold difference in performance overhead, the Principle of Cryptographic Stratification requires that PQC key exchange migration be prioritized above signatures. According to the Footprint-Aware Design Principle, the 1,055% increase in JWT size for Dilithium3 signatures must be accommodated via buffer setup, storage design, and network capacity improvements. The Principle of Agility by Construction requires that abstraction layers, policy enforcement, and negotiation protocols be integrated into API designs from the start rather than retrofitted. The Principle of Hybrid Transition Management recommends adopting hybrid cryptography as a transitory bridge (12–24 months) with defined deprecation timelines to avoid lasting architectural debt. The Principle of Capacity-Aware Migration Planning recommends full-load testing of PQC signatures at peak concurrency, since su-

per-linear degradation was seen beyond beyond 300 concurrent connections (see Figure 4b). The principle of cross-layer impact assessment necessitates a combined evaluation of network, storage, compute, and operational expenses, since increases in signature size have an impact on logging, monitoring, certificate administration, and latency. These six principles are not independent; they support one another, forming a cohesive framework for migration based on real facts rather than theoretical conjecture.

A thorough examination of the study's shortcomings is necessary for scientific integrity and to guide future research. The experimental testbed used uniform, high-performance server hardware (dual Intel Xeon Gold 6248R, 384 GB RAM, NVMe storage), which prioritized internal validity but limited generalizability to resource-constrained contexts like edge devices, IoT nodes, and ARM-based gateways. There was no specific cryptographic accelerator (FPGA, GPU, or HSM), indicating a worst-case CPU-only situation. Energy usage and power efficiency were not monitored, despite the considerable increase in CPU cycles for PQC activities, which has a direct impact on the operating expenses of the data center and its carbon footprint. The tests' time span (hours to days) excludes long-term operational degradation, key rotation effects, and multi-tenant interference in cloud installations. These restrictions do not invalidate the results; rather, they establish the parameters within which they should be evaluated.

Future research should overcome these constraints in multiple interrelated approaches. To begin, multi-platform benchmarking across diverse hardware architectures is required, including resource-constrained edge devices (Raspberry Pi, ARM-based gateways) to simulate IoT scenarios, as well as virtualized cloud instances (AWS EC2, Google Cloud) with different CPU architectures (AMD EPYC, AWS Graviton) to reflect modern cloud-native deployments. Third, longitudinal studies in production contexts are required to confirm the observed performance models and cost-benefit forecasts over multi-year deployment horizons. Experimenting with hardware acceleration for lattice-based operations, such as FPGA or GPU-based polynomial multiplication, might make PQC signatures possible for latency-sensitive applications, according to predictions in this research. Fifth, application-layer cryptographic agility standards and negotiation protocols for APIs and service meshes (apart from TLS) remain unexplored, indicating a serious standardization gap. Sixth, new architectural paradigms such as serverless computing, service mesh architectures, and secret computing environments need dedicated PQC performance and security studies.

The transition to Post-quantum API security is not a single event but a multi-year, multi-phase engineering endeavor. This research has presented an evidence-based roadmap: immediately adopting PQC key exchange for

forward secrecy with little operational effect, concurrent deployment of cryptographic agility frameworks to enable policy-driven migration, use of hybrid cryptography as a temporary bridge (12–18 months) for backward compatibility, and systematic investment in hardware acceleration and capacity planning for PQC signatures. A key takeaway from the research is that PQC migration is best understood as a series of options, each with its own cost-benefit profile. The first choice, relocating key exchange, is low-cost and high-benefit, and it should be implemented quickly. The second option, migrating signatures, is costly and needs careful planning, hardware investment, and even architectural modifications. The final option, whether to adopt hybrid cryptography, is determined by the organization's risk tolerance and the estimated timescale for PQC maturity. This sequential choice paradigm offers practical help to practitioners traversing the challenging landscape of Post-quantum API security.

The quantum danger does not spell the end of digital security, but it does need a deliberate, systematic, and strategic response. The current work adds to this answer by presenting measurable performance statistics, proven architectural patterns, and risk-aware transition techniques. However, we must undertake considerable work, particularly in hardware acceleration, energy efficiency, and standardization. It is intended that this study would serve as a basis for future research and practical guidance for businesses embarking on the difficult but important road toward quantum-resistant API architecture.

List of abbreviations

RFC	Request for Comments.
AES	Advanced Encryption Standard
ANOVA	Analysis of Variance
API	Application Programming Interface
CAL	Cryptographic Abstraction Layer
CCS	Centralized Cryptographic Service
CI	Confidence Interval
CPU	Central Processing Unit
CRQC	Cryptographically Relevant Quantum Computer
CRUD	Create, Read, Update, Delete
DLP	Discrete Logarithm Problem
DSA	Digital Signature Algorithm
DSL	Domain-Specific Language
DSR	Design Science Research
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie-Hellman
ECDHE	Elliptic Curve Diffie-Hellman Ephemeral
ECDSA	Elliptic Curve Digital Signature Algorithm
ECDLP	Elliptic Curve Discrete Logarithm Problem
FFT	Fast Fourier Transform
FIPS	Federal Information Processing Standards
FPGA	Field-Programmable Gate Array
GPU	Graphics Processing Unit

HKDF	HMAC-based Key Derivation Function
HMAC	Hash-based Message Authentication Code
HNDL	Harvest Now, Decrypt Later
HSM	Hardware Security Module
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IETF	Internet Engineering Task Force
IoT	Internet of Things
JSON	JavaScript Object Notation
JWT	JSON Web Token
KDF	Key Derivation Function
KEM	Key Encapsulation Mechanism
LWE	Learning with Errors
MD5	Message Digest Algorithm 5
MER	Migration Effort Reduction
MLWE	Module-Learning with Errors
ML-KEM	Module-Lattice-Based Key Encapsulation Mechanism
ML-DSA	Module-Lattice-Based Digital Signature Algorithm
NIST	National Institute of Standards and Technology
NTRU	Number Theory Research Unit
OOM	Out-of-Memory
OQS	Open Quantum Safe
PEP	Policy Enforcement Point
PKI	Public Key Infrastructure
PQC	Post-Quantum Cryptography
QROM	Quantum Random Oracle Model
QoS	Quality of Service
RAID	Redundant Array of Independent Disks
RAM	Random Access Memory
RAPL	Running Average Power Limit
ROM	Random Oracle Model
RPS	Requests Per Second
RSA	Rivest-Shamir-Adleman
RSS	Resident Set Size
SHA	Secure Hash Algorithm
SIS	Short Integer Solution
SLA	Service-Level Agreement
SSD	Solid-State Drive
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UML	Unified Modeling Language
VLAN	Virtual Local Area Network
X25519	Elliptic Curve Diffie-Hellman using Curve25519
YAML	Yet Another Markup Language
ZFS	Zettabyte File System
0-RTT	Zero Round-Trip Time

Author Contributions

The author (Gabriel Silva Atencio) is solely responsible for Conceptualization, Methodology, Validation, Investigation, Resources, Data curation, Writing - original

draft, Writing - review & editing, Visualization, Project administration,. The author has reviewed and approved the final version of the manuscript.

ORCID Gabriel Silva Atencio: 0000-0002-4881-181X

Availability of Data and Materials

The datasets generated and analyzed during the current study are openly available in the Zenodo repository at <https://doi.org/10.5281/zenodo.23019043> and include the raw performance metrics (latency percentiles P50–P99.9, throughput in requests per second, CPU utilization, memory footprint, network overhead, and storage impact) collected from 180 experimental runs across 6 cryptographic configurations (C1–C6), 2 workload models (Alpha and Beta), and 3 load levels (250, 500, and 750 concurrent users), as described in the Materials and Methods section; the repository also contains cryptographic operation timings for key generation, encapsulation/decapsulation, signing, and verification for each Post-quantum algorithm (Kyber768, Dilithium3, and FALCON-512) alongside their classical baselines (X25519 and ECDSA), the statistical analysis scripts and Jupyter notebooks used to generate all results, figures, and tables reported in this study, and the testbed configuration files required to reproduce the experimental environment, thereby ensuring full reproducibility of the findings and enabling other researchers to validate and extend this work.

Conflicts of Interest

The author declares that he has no conflicts of interest in this work.

AI Declaration

The author used Grammarly for language refinement and grammar checking during the preparation of this manuscript. No AI tool was used for the generation of scientific data, analysis, or conceptualization. The scientific content, interpretation, and conclusions remain solely the responsibility of the authors.

Funding

No external funding was received for this research.

Acknowledgments

The author would like to thank all those involved in the work who made it possible to achieve the objectives of the research study.

References

- [1] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pp. 124–134, 20–22 Nov. 1994 1994, doi: <https://doi.org/10.1109/SFCS.1994.365700>.
- [2] M. Kumar and B. Mondal, "Study on Implementation of Shor's Factorization Algorithm on Quantum Computer," *SN Computer Science*, vol. 5, no. 4, p. 413, 2024/04/08 2024, doi: <https://doi.org/10.1007/s42979-024-02771-y>.
- [3] G. Alagic et al., "Status Report on the Fourth Round of the NIST Post-Quantum Cryptography Standardization Process," National Institute of Standards and Technology: Gaithersburg, MD, USA, 2025, doi: <https://doi.org/10.6028/NIST.IR.8545>.
- [4] R. Hai, S.-H. Hung, T. Coopmans, T. Littau, and F. Geerts, "Quantum data management in the NISQ era," *Proceedings of the VLDB Endowment*, vol. 18, no. 6, pp. 1720–1729, 2025, doi: <https://doi.org/10.14778/3725688.3725701>.
- [5] D. J. Bernstein, "Post-quantum Cryptography," *Encyclopedia of Cryptography, Security and Privacy*, pp. 1846–1847, 2025, doi: https://doi.org/10.1007/978-3-030-71522-9_386.
- [6] N. Sanger and S. Abeck, "Fine-Grained Authorization in Microservice Architecture: A Decentralized Approach," *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing*, pp. 1219–1222, 2024, doi: <https://doi.org/10.1145/3605098.3636121>.
- [7] A. Nair, S. Nicolazzo, A. Nocera, A. R. R. K., and P. V., "Sok: Zero Trust as a Strategy to Address Devsecops Challenges," *2025 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pp. 546–554, 30 June–4 July 2025 2025, doi: <https://doi.org/10.1109/EuroSPW67616.2025.00069>.
- [8] L. Marchesi, M. Marchesi, and R. Tonelli, "A survey on Cryptoagility and Agile Practices in the light of quantum resistance," *Information and Software Technology*, vol. 178, p. 107604, 2025/02/01/ 2025, doi: <https://doi.org/10.1016/j.infsof.2024.107604>.
- [9] J. A. Montenegro, R. Rios, and J. Lopez-Cerezo, "A performance evaluation framework for post-quantum TLS," *Future Generation Computer Systems*, vol. 175, p. 108062, 2026/02/01/ 2026,

- doi: <https://doi.org/10.1016/j.future.2025.108062>.
- [10] R. Rios, J. A. Montenegro, A. Muoz, and D. Ferraris, "Toward the Quantum-Safe Web: Benchmarking Post-Quantum TLS," *IEEE Network*, vol. 39, no. 5, pp. 247–253, 2025, doi: <https://doi.org/10.1109/MNET.2025.3531116>.
 - [11] D. Agarwal, R. Arul, and K. Thirugnanasambandam, "Review on the Implications of Crystals Kyber in LTE Networks," *Security and Privacy*, vol. 2799, pp. 19–37, 2026, doi: https://doi.org/10.1007/978-3-032-12834-8_2.
 - [12] A. Kerimbayeva *et al.*, "A Lightweight Variant of Falcon for Efficient Post-Quantum Digital Signature," *Information*, vol. 16, no. 7, p. 564, 2025, doi: <https://doi.org/10.3390/info16070564>.
 - [13] G. Olaoye, "Post-Quantum Cryptography: Evaluating and Standardizing Quantum-Resistant Algorithms," *Authorea*, 2025, doi: <https://doi.org/10.22541/au.174422820.03426084/v1>.
 - [14] R. A. Santa Quintero and S. L. Meneses Velandia, "Integrating Post-Quantum Cryptography in Document Lifecycle Management: Harmonizing ISO 15489-1 and NIST PQC Standards," *Proceedings of the 2025 8th International Conference on Systems Engineering-Cybersecurity & AI: Building a reliable digital future*, pp. 47–52, 2025, doi: <https://doi.org/10.1145/3771678.3771685>.
 - [15] T. L. Astrizi and R. Custódio "Seamless Transition to Post-Quantum TLS 1.3: A Hybrid Approach Using Identity-Based Encryption," *Sensors*, vol. 24, no. 22, p. 7300, 2024, doi: <https://doi.org/10.3390/s24227300>.
 - [16] A. Mohammed, N. U. Mohammed, S. K. R. Gunda, and Z. Mohammed, "Fundamental Principles of Network Security," *Journal of Theory and Practice of Management Science*, vol. 5, no. 5, 2025, doi: [https://doi.org/10.53469/jtpms.2025.05\(5\).06](https://doi.org/10.53469/jtpms.2025.05(5).06).
 - [17] Z. Zhang and Y. Zhao, "Enhanced Elliptic Curve Cryptography (EECC)," *Procedia Computer Science*, vol. 247, pp. 1324–1330, 2024/01/01/ 2024, doi: <https://doi.org/10.1016/j.procs.2024.10.158>.
 - [18] Y. Sun and L.-A. Wu, "Quantum search algorithm on weighted databases," *Scientific Reports*, vol. 14, no. 1, p. 30169, 2024/12/04 2024, doi: <https://doi.org/10.1038/s41598-024-81701-7>.
 - [19] M. Joshi, M. K. Mishra, and S. Karthikeyan, "Leveraging Grover's Algorithm for Quantum Searchable Encryption in Cloud Infrastructure and its application in AES Resource Estimation," *International Journal of Theoretical Physics*, vol. 63, no. 8, p. 209, 2024/08/21 2024, doi: <https://doi.org/10.1007/s10773-024-05751-3>.
 - [20] M. Kundalakesi and M. R. Devi, "Post-Quantum Cryptography Methods," *Quantum computing and artificial intelligence: the industry use cases*, pp. 77–97, 2025, doi: <https://doi.org/10.1002/9781394242399.ch2>.
 - [21] D. Dziechciarz and M. Niemiec, "Efficiency analysis of NIST-standardized post-quantum cryptographic algorithms for digital signatures in various environments," *Electronics*, vol. 14, no. 1, p. 70, 2024, doi: <https://doi.org/10.3390/electronics14010070>.
 - [22] L. Li, X. Lu, and K. Wang, "Hash-based signature revisited," *Cybersecurity*, vol. 5, no. 1, p. 13, 2022/07/01 2022, doi: <https://doi.org/10.1186/s42400-022-00117-w>.
 - [23] K. Cherkaoui Dekkaki, I. Tasic, and M.-D. Cano, "Exploring post-quantum cryptography: Review and directions for the transition process," *Technologies*, vol. 12, no. 12, p. 241, 2024, doi: <https://doi.org/10.3390/technologies12120241>.
 - [24] A. Margara, G. Cugola, N. Felicioni, and S. Cilloni, "A model and survey of distributed data-intensive systems," *ACM Computing Surveys*, vol. 56, no. 1, pp. 1–69, 2023, doi: <https://doi.org/10.1145/3604801>.
 - [25] M. Yaseen, M. B. Alshawki, and P. Ligeti, "Security Analysis and Deployment Measurement of Transport Layer Security Protocol," pp. 725–739, 2022, doi: https://doi.org/10.1007/978-981-16-8892-8_55.
 - [26] J. Singh and N. K. Chaudhary, "OAuth 2.0 : Architectural design augmentation for mitigation of common security vulnerabilities," *Journal of Information Security and Applications*, vol. 65, p. 103091, 2022/03/01/ 2022, doi: <https://doi.org/10.1016/j.jisa.2021.103091>.
 - [27] I. Zulkarneev and K. A. Basalay, "JSON Web Tokens Lifecycle-Based Threat Classification," *2024 IEEE 25th International Conference of Young Professionals in Electron Devices and Materials (EDM)*, pp. 1920–1924, 28 June–2 July 2024 2024, doi: <https://doi.org/10.1109/EDM61683.2024.10615042>.
 - [28] P. Scalise, R. Garcia, M. Boeding, M. Hempel, and H. Sharif, "An Applied Analysis of Securing 5G/6G Core Networks with Post-Quantum Key Encapsulation Methods," *Electronics*, vol. 13, no. 21, p. 4258, 2024, doi: <https://doi.org/10.3390/electronics13214258>.
 - [29] I. W. R. Pinastawa, M. G. Pradana, and N. Maulana, "Web Server Performance Evaluation: Comparing Nginx and Apache Using K6 Testing Methods for Load, Spike, Soak, and Performance," *2024 International Conference on Informatics, Multimedia, Cyber and Information System*

- (ICIMCIS), pp. 1002–1007, 20–21 Nov. 2024 2024, doi: <https://doi.org/10.1109/ICIMCIS63449.2024.10956402>.
- [30] E. dos Santos and S. I. Casas, "A catalog of API Gateway metrics and its quantitative evaluation," *DYNA: revista de la Facultad de Minas. Universidad Nacional de Colombia. Sede Medellin*, vol. 92, no. 237, pp. 106–114, 2025, doi: <https://doi.org/10.15446/dyna.v92n237.118741>.
 - [31] G. Hanaoka, T. Matsuda, and J. C. N. Schuldt, "A New Combiner for Key Encapsulation Mechanisms," *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 102, no. 12, pp. 1668–1675, 2019, doi: <https://doi.org/10.1587/transfun.E102.A.1668>.
 - [32] M. Backendal, S. Clermont, M. Fischlin, and F. Günther, "Key derivation functions without a grain of salt," *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 393–426, 2025, doi: https://doi.org/10.1007/978-3-031-91101-9_14.
 - [33] J. Chen, W. Peng, Y. Wang, and Y. Bian, "On the Security and Efficiency of TLS 1.3 Handshake with Hybrid Key Exchange from CPA-Secure KEMs," *Entropy*, vol. 27, no. 12, p. 1242, 2025, doi: <https://doi.org/10.3390/e27121242>.
 - [34] B. Blanchet and C. Jacomme, "Post-quantum sound CryptoVerif and verification of hybrid TLS and SSH key-exchanges," *2024 IEEE 37th Computer Security Foundations Symposium (CSF)*, pp. 543–556, 2024, doi: <https://doi.org/10.1109/CSF61375.2024.00050>.