GUIDE

Meta's Post-Quantum Cryptography (PQC) Migration Guide

Migration: Classical Cryptography (e.g., RSA, ECC) → NIST PQC Standards (e.g., CRYSTALS-Kyber, CRYSTALS-Dilithium) Effort estimate: Months to years, ongoing for large organizations **Complexity:** COMPLETE-REWRITE

Breaking changes in this upgrade:

- Compatibility-breaking changes to encryption protocols and data formats.
- High-risk certificate infrastructure changes affecting trust chains system-wide.
- Fundamental shifts in cryptographic primitives requiring extensive code and system updates.

Why PQC Migration is Critical (The Quantum Threat)

The digital world relies heavily on classical public-key cryptography, such as RSA and Elliptic Curve Cryptography (ECC), to secure communications and data. However, the advent of sufficiently powerful quantum computers poses an existential threat to these foundational cryptographic systems. Algorithms like Shor's algorithm can efficiently break RSA and ECC, while Grover's algorithm can significantly reduce the security of symmetric ciphers and hash functions.

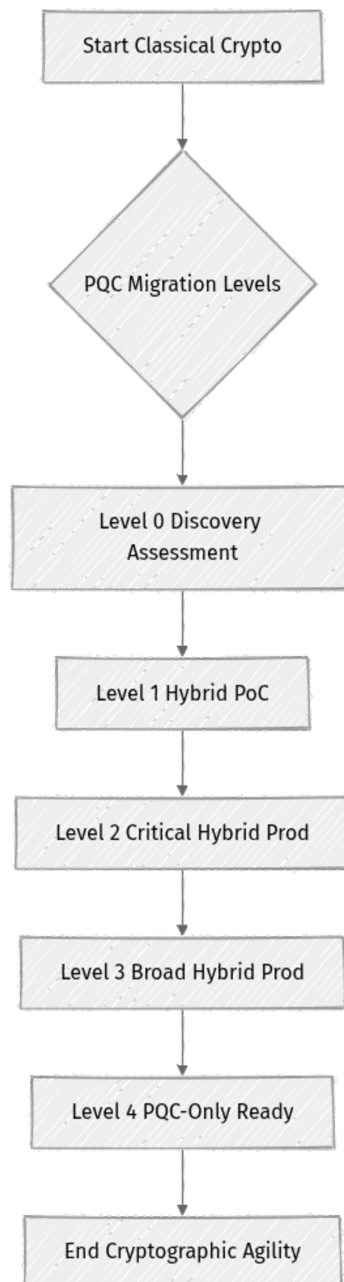
This isn't a distant future problem; the "harvest now, decrypt later" threat is already here. Adversaries could be collecting encrypted data today, intending to decrypt it once quantum computers become available. For large organizations like Meta, with vast amounts of sensitive user data and critical infrastructure, proactive migration to Post-Quantum Cryptography (PQC) is not merely a best practice—it's an urgent imperative to ensure long-term security and trust.

Meta's PQC Migration Levels Framework

Meta has developed a structured "PQC Migration Levels" framework to manage the immense complexity of this transition. This framework categorizes cryptographic assets and systems based on their criticality, exposure, and the effort required for migration, allowing for a phased and prioritized approach. While the exact number of levels and their precise definitions may vary, a typical framework might include:

- **Level 0: Discovery & Assessment:** Initial inventory of all cryptographic assets, dependencies, and threat modeling. No PQC implementation yet.
- **Level 1: PQC Readiness & Hybrid Proof-of-Concept:** Evaluation of PQC algorithms, integration of PQC libraries, and small-scale, non-production hybrid deployments (classical + PQC) for testing.
- **Level 2: Hybrid Production Deployment (Critical Systems):** Phased rollout of hybrid PQC to the most critical, high-risk systems (e.g., TLS for key services, critical internal communications). Classical algorithms are still present but PQC is active.
- **Level 3: Broad Hybrid Production Deployment:** Expansion of hybrid PQC across a wider range of services and applications, including less critical but still important systems.
- **Level 4: PQC-Only Readiness (Future State):** Systems are fully capable of operating in PQC-only mode, ready for a future where classical cryptography is entirely deprecated. This requires robust cryptographic agility.

This structured approach ensures that resources are allocated effectively, risks are managed incrementally, and the organization can adapt to the evolving PQC landscape.



Meta's Strategic Approach: Hybrid-First and Cryptographic Agility

Recognizing the inherent risks and uncertainties of a complete cryptographic overhaul, Meta has adopted a "hybrid-first" strategy. This means that during the transition, systems will simultaneously support both classical (e.g., ECC) and PQC (e.g., Kyber) algorithms. This approach offers several critical advantages:

- **Risk Mitigation:** It provides a safety net. If a PQC algorithm is found to be insecure or has performance issues, the classical algorithm can still provide security.

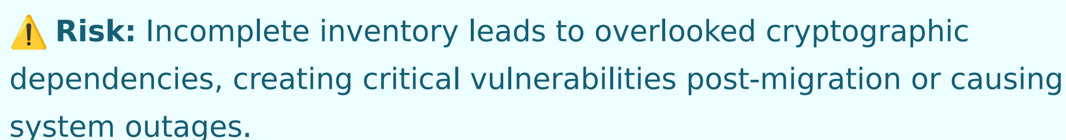
- **Interoperability:** It allows for a gradual rollout, ensuring compatibility with systems that have not yet migrated to PQC.
- **Testing and Validation:** It enables real-world testing of PQC algorithms in production environments without immediately compromising security.

Alongside the hybrid approach, Meta emphasizes "cryptographic agility." This principle ensures that systems are designed to easily swap out cryptographic primitives and algorithms without requiring a full re-architecture. This is vital because the PQC landscape is still evolving, and new algorithms may be selected by NIST, or existing ones may be broken. Building agility into the core infrastructure allows for rapid adaptation to future cryptographic changes.

Risk Assessment and Prioritization of Cryptographic Assets

A successful PQC migration begins with a thorough understanding of an organization's cryptographic attack surface. This involves:

1. **Asset Identification:** Cataloging all systems, applications, data stores, and communication channels that rely on cryptography. This includes identifying where keys are stored, how they are managed, and which algorithms are in use.
2. **Data Classification:** Determining the sensitivity and longevity requirements of data protected by cryptography. Data that needs to remain confidential for decades (e.g., personal identifiers, trade secrets) is a higher priority for PQC migration due to the "harvest now, decrypt later" threat.
3. **Threat Modeling:** Assessing the likelihood and impact of quantum attacks on identified assets. This helps prioritize which systems require PQC protection first.
4. **Dependency Mapping:** Understanding the intricate web of dependencies between cryptographic services, libraries, and applications. A change in one component can have cascading effects.



⚠️ **Risk:** Incomplete inventory leads to overlooked cryptographic dependencies, creating critical vulnerabilities post-migration or causing system outages.

Impact Analysis: Breaking Changes and Deprecated Algorithms

The transition to PQC is not a simple patch; it represents a fundamental shift in cryptographic primitives with significant breaking changes. Understanding these impacts is crucial for planning.

Compatibility-Breaking Changes to Encryption Protocols and Data Formats

PQC algorithms, particularly Key Encapsulation Mechanisms (KEMs) like CRYSTALS-Kyber, often produce larger ciphertexts and public keys compared to their classical counterparts. Similarly, signature schemes like CRYSTALS-Dilithium can generate larger signatures. This directly impacts existing network protocols and data storage formats.

- **What changed:** Increased size of cryptographic objects (keys, ciphertexts, signatures) requires adjustments to message buffers, database schema, and network packet sizes.
- **Before/After Code Example (Conceptual):**

  Before (Classical Protocol Message Structure):

```
    {
      "header": "...",
      "payload": "...",
      "encryption_key_id": "rsa-key-123",
      "encrypted_data": "base64(RSA_AES_Encrypted_Data)",
      "signature": "base64(RSA_Signature_Small)"
    }
```

```
**After (PQC Hybrid Protocol Message Structure):**
```

```
    {
      "header": "...",
      "payload": "...",
      "encryption_key_id": "hybrid-key-456",
      "pqc_ciphertext": "base64(Kyber_Ciphertext_Larger)",
      "classical_ciphertext": "base64(ECC_AES_Encrypted_Data)",
      "pqc_signature": "base64(Dilithium_Signature_Larger)",
      "classical_signature": "base64(ECC_Signature_Small)"
    }
```

```
- **Impact:** Existing parsers might fail, database fields might truncate data, and network performance could be affected by larger payloads.
```

High-Risk Certificate Infrastructure Changes Affecting Trust Chains System-Wide

Public Key Infrastructure (PKI) is the backbone of trust in digital systems. Migrating PKI to support PQC is one of the most complex and high-risk aspects of the transition.

- **What changed:** Certificate Authorities (CAs) must be updated to issue PQC certificates (e.g., X.509 certificates with PQC public keys and signatures). Client-side trust stores and validation logic must also be updated to recognize and validate these new certificate types.
- **Before/After Code Example (Conceptual):**

  Before (Classical Certificate Generation Request):

```
# Request for an ECC certificate
openssl req -new -newkey ec -pkeyopt ec_paramgen_curve:prime256v1 \
  -nodes -keyout classical_key.pem -out classical_cert.csr
```

```
**After (PQC Hybrid Certificate Generation Request - conceptual, requires PQC-enabled OpenSSL/library):**
```

```
# Request for a hybrid certificate (e.g., ECC + Dilithium)
# This might involve custom extensions or a new standard for hybrid certs.
# Actual command would depend on PQC library integration with OpenSSL.
openssl req -new -newkey hybrid -pkeyopt hybrid_schemes:p256_dilithium3 \
  -nodes -keyout hybrid_key.pem -out hybrid_cert.csr
```

```
- **Impact:** A poorly executed PKI migration can break trust chains, rendering entire systems inaccessible or insecure. All clients and services relying on certificates must be updated.
```

Fundamental Shifts in Cryptographic Primitives Requiring Extensive Code and System Updates

Developers accustomed to using established APIs for RSA or ECC will need to adopt new PQC-specific APIs. This isn't just a library update; it's a change in how cryptographic operations are performed.

- **What changed:** Direct replacement of classical cryptographic API calls with new PQC API calls for key generation, encryption/decryption, and signing/verification.
- **Before/After Code Example (Conceptual - using a generic crypto library interface):**

  Before (Classical Key Exchange and Encryption):

```
    from classical_crypto import RSA_Encrypt, ECC_KeyExchange

    # RSA encryption
    public_key_rsa = get_rsa_public_key()
    ciphertext = RSA_Encrypt(public_key_rsa, "sensitive data")

    # ECC key exchange
    local_private, local_public = ECC_KeyExchange.generate_keys()
    shared_secret = ECC_KeyExchange.derive_shared_secret(local_private, remote
_public)
```

```
**After (PQC Hybrid Key Exchange and Encryption):**
```

```
    from pqc_hybrid_crypto import Kyber_KEM, Dilithium_Sign, AES_GCM

    # Kyber KEM (Key Encapsulation Mechanism)
    pqc_public_key = get_kyber_public_key()
    ciphertext_pqc, shared_secret_pqc = Kyber_KEM.encapsulate(pqc_public_key)

    # Use shared_secret_pqc for symmetric encryption
    encrypted_data = AES_GCM.encrypt(shared_secret_pqc, "sensitive data")

    # Dilithium signing
    pqc_private_key = get_dilithium_private_key()
    signature_pqc = Dilithium_Sign(pqc_private_key, "message hash")
```

```
- **Impact:** Requires significant code changes across all applications using
cryptography, re-training developers, and extensive testing for correctness
and security.
```

Removed/Deprecated Algorithms

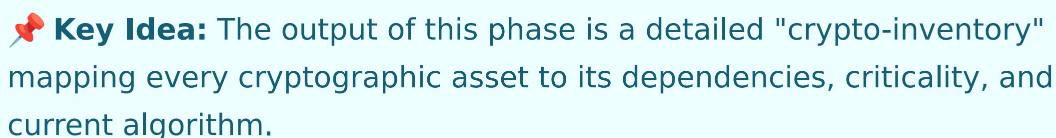
Classical cryptographic algorithms (e.g., RSA, ECC) will be deprecated or removed for quantum-safe contexts. While they may remain for backward compatibility in hybrid modes, their use for new quantum-vulnerable applications will be discouraged or forbidden.

- **What changed:** Existing code relying solely on RSA or ECC for quantum-vulnerable functions will need to be rewritten to use PQC algorithms.
- **Impact:** Forces a complete rewrite of cryptographic modules in legacy systems.

Discovery and Inventory of Cryptographic Dependencies

Before any code changes, a comprehensive discovery phase is paramount. This involves:

1. **Automated Scanning:** Utilizing static and dynamic analysis tools to scan codebases for cryptographic API calls, key sizes, and algorithm identifiers.
2. **Configuration Analysis:** Reviewing configuration files, infrastructure-as-code, and deployment manifests to identify cryptographic parameters.
3. **Network Traffic Analysis:** Monitoring network traffic to identify protocols and algorithms in use (e.g., TLS versions, cipher suites).
4. **Documentation Review:** Consulting existing architecture diagrams, design documents, and security policies.
5. **Interviewing Teams:** Engaging with development, operations, and security teams to understand their use of cryptography.



> 📌 **Key Idea:** The output of this phase is a detailed "crypto-inventory" mapping every cryptographic asset to its dependencies, criticality, and current algorithm.

Planning and Algorithm Selection (NIST Standards)

With a comprehensive inventory, the next step is strategic planning and algorithm selection.

1. **NIST Algorithm Selection:** Focus on the algorithms selected by NIST for standardization. Currently, CRYSTALS-Kyber for Key Encapsulation Mechanisms (KEMs) and CRYSTALS-Dilithium for digital signatures are primary choices.
2. **Performance Benchmarking:** Evaluate the performance characteristics (CPU, memory, key/signature/ciphertext size, latency) of selected PQC algorithms in your specific environment. PQC algorithms often have larger parameters and can be slower than classical counterparts.
3. **Security Level Alignment:** Map the security levels of PQC algorithms to the required security postures of your systems.
4. **Migration Wave Planning:** Define phased migration waves based on the risk assessment and dependency mapping. Start with less critical systems or internal services to gain experience, then move to customer-facing or high-value assets.

Implementation and Integration Strategies

Implementation is where the code changes begin.

1. **Library Integration:** Integrate PQC-enabled cryptographic libraries (e.g., Open Quantum Safe (OQS) liboqs, or Meta's internal PQC libraries) into your application stack.
2. **Hybrid Mode Development:** Implement the hybrid approach by enabling systems to negotiate and use both classical and PQC algorithms simultaneously. This typically involves:
    - **Dual Key Pairs:** Generating both classical and PQC key pairs for identities.
    - **Hybrid Key Exchange:** Combining classical and PQC KEMs to derive a shared secret.
    - **Hybrid Signatures:** Signing data with both classical and PQC signature schemes.

3. **Protocol Updates:** Modify existing protocols (e.g., TLS, SSH, custom RPC protocols) to support PQC negotiation and larger PQC message formats.
4. **Key Management System (KMS) Updates:** Ensure your KMS can generate, store, and manage PQC keys. This includes key rotation policies.

> ⚡ **Real-world insight:** Meta often uses custom extensions to existing protocols (like TLS) to introduce PQC in a hybrid fashion, allowing for granular control and phased rollouts.

Deployment, Monitoring, and Guardrails

Deployment must be cautious and well-monitored.

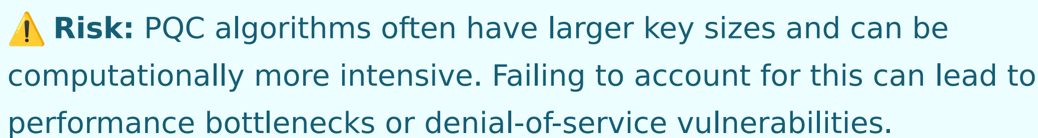
1. **Phased Rollout:** Deploy PQC capabilities in small, controlled increments. Start with internal testing environments, then move to canary deployments, and finally to full production.
2. **Performance Monitoring:** Continuously monitor key metrics such as CPU utilization, memory consumption, network bandwidth, and latency. PQC algorithms can introduce overhead.
3. **Error Logging and Alerting:** Implement robust logging for cryptographic operations and set up alerts for any PQC-related failures, interoperability issues, or performance degradations.
4. **Guardrails:** Establish automated checks and circuit breakers that can detect issues and revert to classical cryptography or a known-good state if problems arise.
5. **Backward Compatibility:** Ensure that systems that have not yet migrated can still communicate with PQC-enabled systems, typically through the hybrid mode.

Testing, Validation, and Performance Considerations

Rigorous testing is non-negotiable for PQC migration.

1. **Unit and Integration Testing:** Verify that individual PQC primitives and their integration points work correctly.

2. **Interoperability Testing:** Crucially, test communication between PQC-enabled systems and classical systems, as well as between different PQC implementations.
3. **Performance Benchmarking:** Conduct extensive performance tests to measure the impact of PQC on:
    - **Latency:** How much longer do PQC key exchanges or signature verifications take?
    - **Throughput:** How many operations per second can the system handle?
    - **Resource Usage:** CPU, memory, and network bandwidth consumption.
    - **Key/Signature/Ciphertext Sizes:** Verify that data formats and storage can accommodate larger PQC objects.
4. **Security Audits:** Conduct independent security audits and penetration testing to identify any vulnerabilities introduced during the migration.
5. **Fuzz Testing:** Subject PQC implementations to fuzz testing to uncover unexpected behavior or vulnerabilities.



⚠️ **Risk:** PQC algorithms often have larger key sizes and can be computationally more intensive. Failing to account for this can lead to performance bottlenecks or denial-of-service vulnerabilities.

Lessons Learned and Practical Takeaways

Meta's journey provides valuable insights for other organizations:

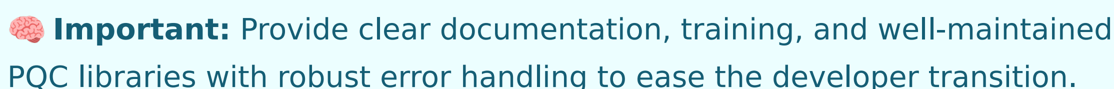
- **Complexity is Underestimated:** PQC migration is a far-reaching transformation, impacting infrastructure, standards, and engineering practices across the entire organization. It's not just a crypto library update.
- **Hybrid-First is Key:** The hybrid approach is essential for managing risk and ensuring backward compatibility during the transition period.
- **Tooling is Critical:** Manual code changes are infeasible at scale. Automated tools, custom scripts, and codemods are indispensable for large-scale modifications.

- **Cryptographic Agility is a Must-Have:** Design systems from the outset to be easily adaptable to new cryptographic algorithms, as the PQC landscape will continue to evolve.
- **Developer Education:** Developers need to understand the new primitives and the implications of PQC.

Addressing Developer Experience Challenges

Migrating to PQC introduces several challenges for developers:

- **New API Paradigms:** PQC primitives (e.g., KEMs vs. traditional asymmetric encryption) require a different mental model and new API calls.
- **Performance Trade-offs:** Developers must be aware of potential performance impacts (larger data sizes, higher CPU usage) and design accordingly.
- **Interoperability Issues:** Ensuring seamless communication between classical, hybrid, and PQC-only components can be complex.
- **Debugging Complexity:** Diagnosing issues in hybrid cryptographic protocols requires deep understanding of both classical and PQC components.
- **Multithreading Issues:** As noted in Meta's experience, integrating new PQC libraries (like liboqs) can introduce subtle multithreading bugs (e.g., segmentation faults), requiring careful handling and fixes.



🧠 **Important:** Provide clear documentation, training, and well-maintained PQC libraries with robust error handling to ease the developer transition.

Leveraging Automated Tools and Internal Support

Given the "complete-rewrite" complexity and "months to years" effort, automation is non-negotiable.

- **Code Modification Tools (Codemods):** Develop or leverage tools that can automatically identify and refactor cryptographic API calls in large codebases.

```
    # Example: Pseudo-command for running a PQC codemod
    # This tool would scan for classical crypto calls and suggest/apply PQC
replacements.
    ./meta-pqc-codemod --config ./pqc_migration_rules.json --target-dir src/
```

```
crypto_modules/ --apply
```

- **Configuration Management Scripts:** Use scripts to update configuration files across thousands of servers to enable PQC cipher suites or protocols.

```
    # Example: Deploying updated server configurations
    ansible-playbook -i inventory/prod.yml playbooks/pqc_server_config.yml --
limit web_servers
```

- **Automated Testing Frameworks:** Build robust, automated test suites that can run PQC-specific tests at scale.

```
    # Example: Running PQC integration tests in CI/CD
    docker exec my_ci_runner ./run_pqc_tests.sh --suite integration --
environment production_canary
```

- **Internal Support Teams:** Establish dedicated security and infrastructure teams to provide expert guidance, troubleshoot issues, and maintain PQC libraries and tools.

Rollback and Contingency Planning

Despite meticulous planning, issues can arise. A well-defined rollback strategy is critical.

Rollback Possible: Yes

Rollback Steps:

1. **Identify Trigger:** Determine clear criteria for initiating a rollback (e.g., severe performance degradation, widespread interoperability failures, critical security vulnerabilities).
2. **Revert Code Changes:**
    - For applications, revert to the previous code version that uses classical cryptography.

```
        # Example: Revert Git commit for a specific service
        git revert <PQC_MIGRATION_COMMIT_HASH> --no-edit
        git push origin main
```

```
- For infrastructure-as-code, revert to the previous configuration.
```

1. **Redeploy Previous Configuration/Binaries:** Deploy the classical-only or previous hybrid version of the affected services.

```
    # Example: Redeploying a service to a previous version
    kubectl rollout undo deployment/my-pqc-service --to-revision=<PREVIOUS_GOO
D_REVISION>
```

1. **Revert Certificate Infrastructure (if applicable):** If PKI changes were deployed, be prepared to revert to classical certificate issuance and validation, or to temporarily disable PQC certificate paths.
    - This is typically the most complex part of a rollback and requires careful pre-planning to ensure classical CAs and validation services remain operational.
2. **Monitor Rollback:** Closely monitor systems during and after the rollback to ensure stability and functionality are restored.
3. **Post-Mortem Analysis:** Conduct a thorough investigation to understand why the rollback was necessary and what lessons can be learned for future migration attempts.

> ✅ **Safe to skip if:** The PQC changes are deployed in a purely additive, non-breaking hybrid mode where classical paths remain fully functional and preferred by default in case of issues. However, for true "complete-rewrite" scenarios, a full rollback plan is essential.

Future-Proofing: Maintaining Cryptographic Agility

The PQC migration is not a one-time event but the beginning of an ongoing commitment to cryptographic agility.

- **Modular Cryptographic Design:** Design systems with clear interfaces for cryptographic operations, allowing for easy swapping of algorithms and libraries without disrupting core application logic.
- **Automated Algorithm Updates:** Invest in tooling and processes that can facilitate rapid updates to cryptographic algorithms in response to new NIST standards or emerging threats.

- **Continuous Monitoring:** Maintain continuous vigilance over the cryptographic landscape, including advances in quantum computing and new cryptanalytic attacks.
- **Research and Development:** Stay engaged with academic and industry research in cryptography to anticipate future needs and challenges.

By embracing cryptographic agility, Meta aims not just to survive the quantum threat but to build a resilient and adaptable security posture for decades to come. This ongoing effort ensures that the organization can respond effectively to the inevitable evolution of cryptographic science and the threat landscape.