CANONICAL LIVING OVERVIEW GUIDE

Post-Quantum Cryptography Implementation and Migration

A post-quantum implementation and migration guide covering quantum risk, cryptographic discovery, FIPS 203/204/205, KEM use, signatures, hybrid protocols, PKI, code signing, data protection, performance, interoperability, testing, vendors, governance, and phased enterprise transition.

PERMANENT ID
MYTHOS-FG-SEC-0017

PUBLICATION CLASS
FIELD GUIDE

VERSION / STATUS
1.0.0-candidate / CANDIDATE

PERMANENT PUBLICATION IDENTITY

Post-Quantum Cryptography Implementation and Migration

Document ID
MYTHOS-FG-SEC-0017

Version
1.0.0-candidate

Status
Candidate Focused Field Guide

Review date
2026-10-26

Publication position
This is a living field guide. It is designed for frequent revision while retaining a stable permanent identity. Candidate status means the content is ready for structured use and review but is not represented as an external certification or authority publication.

LEVEL	CLASS	CADENCE	EVIDENCE
L1-L4	FIELD GUIDE	QUARTERLY	SOURCE-BOUND
Foundational through frontier	Practical and educational	Interim updates as needed	Limitations remain visible

Reader orientation

Dimension	Engineering position
Primary domain	MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography
Secondary domain	MYTHOS-QTC; MYTHOS-NET; MYTHOS-SWE
Audience	Security, cryptography, PKI, network, software, cloud, hardware, product, procurement, and risk teams.
Prerequisites	Applied cryptography, PKI, key management, protocols, software deployment, and enterprise architecture fundamentals.
Publisher / owner	Mythos Systems / Aaron Stovall
Public classification	Public technical guidance

Expected outcomes

Identify quantum-vulnerable algorithms, systems, data, and dependencies across the enterprise.

Implement ML-KEM, ML-DSA, and SLH-DSA according to their standardized interfaces and security properties.

Use KEMs and hybrid mechanisms safely in protocols without treating algorithm replacement as a drop-in edit.

Plan migration across TLS, VPN, PKI, signing, firmware, data, suppliers, and constrained systems.

Validate performance, size, interoperability, failure, rollback, and long-term crypto-agility.

How to use this guide

Read the guide as a progressive system rather than a disconnected encyclopedia. Foundational material establishes the mental model; practitioner sections convert it into repeatable implementation and operations; advanced sections expose architecture, scale, security, and failure behavior; frontier sections describe technologies whose evidence or maturity is still evolving.

Suggested reading path
New practitioners should follow the chapters in order and complete the laboratories. Experienced engineers may begin with the architecture, failure, validation, or frontier sections, then use the related-document map to deepen a specific topic.

Learning and assurance model

Level	Reader outcome
L1 - Foundational	Terminology, first principles, component roles, packet or state flow, and simple working examples.
L2 - Practitioner	Implementation, configuration, automation, testing, troubleshooting, and operational evidence.
L3 - Advanced	Architecture, scale, concurrency, resilience, threat models, performance, and complex migrations.
L4 - Frontier	Emerging systems, research directions, technical uncertainty, readiness gates, and adoption signals.

Contents

CHAPTER 01

Quantum threat, timelines, and migration urgency

Translate quantum computing risk into data- and system-specific engineering decisions.

Chapter operating position
Translate quantum computing risk into data- and system-specific engineering decisions.

1.1 Quantum impact on classical cryptography

Quantum impact on classical cryptography must be implemented as an observable engineering mechanism, not a product label or maturity claim. Understand Shor-related risk to factoring and discrete logarithms and Grover-related implications for symmetric parameters. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the quantum risk, inventory, standardized algorithm, protocol integration, performance, migration, verification, and retirement chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for quantum impact on classical cryptography.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating quantum impact on classical cryptography as a product feature instead of an end-to-end engineering system.
- Relying on intended policy, rule code, or controller state without proving applied and operational behavior.
- Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for quantum impact on classical cryptography.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

1.2 Harvest-now-decrypt-later

Harvest-now-decrypt-later must be implemented as an observable engineering mechanism, not a product label or maturity claim. Identify data whose confidentiality must survive beyond the plausible migration and cryptanalytic horizon. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the quantum risk, inventory, standardized algorithm, protocol integration, performance, migration, verification, and retirement chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for harvest-now-decrypt-later.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Implementation sketch

```
pqc_migration:
  system: external-api-gateway
  current:
    key_establishment: ecdhe
    signatures: ecdsa
  target:
    key_establishment: hybrid-classical-ml-kem
    signatures: staged-ml-dsa
  gates:
    - official-standard-and-library-support
    - interoperability-matrix-pass
    - mtu-and-fragmentation-pass
    - latency-and-capacity-pass
    - downgrade-protection-pass
    - rollback-tested
  inventory_evidence:
    protocols: [tls, vpn, certificates, hsm, code-signing]
```

Failure patterns

Treating harvest-now-decrypt-later as a product feature instead of an end-to-end engineering system.

Relying on intended policy, rule code, or controller state without proving applied and operational behavior.

Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for harvest-now-decrypt-later.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

1.3 System and signature longevity

System and signature longevity must be implemented as an observable engineering mechanism, not a product label or maturity claim. Assess firmware, documents, software updates, identity, certificates, archives, infrastructure, and safety systems. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the quantum risk, inventory, standardized algorithm, protocol integration, performance, migration, verification, and retirement chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric

paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for system and signature longevity.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating system and signature longevity as a product feature instead of an end-to-end engineering system.
- Relying on intended policy, rule code, or controller state without proving applied and operational behavior.
- Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for system and signature longevity.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

1.4 Uncertainty and decision making

Uncertainty and decision making must be implemented as an observable engineering mechanism, not a product label or maturity claim. Avoid precise quantum dates; use consequence, data lifetime, migration lead time, dependency, and reversibility. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the quantum risk, inventory, standardized algorithm, protocol integration, performance, migration, verification, and retirement chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for uncertainty and decision making.

Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.

Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.

Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.

Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.

Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating uncertainty and decision making as a product feature instead of an end-to-end engineering system.

Relying on intended policy, rule code, or controller state without proving applied and operational behavior.

Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for uncertainty and decision making.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

Chapter 01 laboratory and review

Practical laboratory
Build a quantum-risk register for ten systems and rank them by data lifetime, trust lifetime, migration complexity, and consequence.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Extend risk modeling to cryptographically relevant quantum computers, distributed attacks, and quantum-safe hardware roots.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 02

Cryptographic discovery and dependency mapping

Find where vulnerable cryptography is used before selecting replacements.

Chapter operating position
Find where vulnerable cryptography is used before selecting replacements.

2.1 Algorithms, protocols, and libraries

Algorithms, protocols, and libraries must be implemented as an observable engineering mechanism, not a product label or maturity claim. Inventory RSA, finite-field DH, ECC, certificates, TLS, SSH, IPsec, email, code signing, firmware, tokens, and proprietary protocols. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the quantum risk, inventory, standardized algorithm, protocol integration, performance, migration, verification, and retirement chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for algorithms, protocols, and libraries.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating algorithms, protocols, and libraries as a product feature instead of an end-to-end engineering system.
- Relying on intended policy, rule code, or controller state without proving applied and operational behavior.
- Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for algorithms, protocols, and libraries.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

2.2 Keys, certificates, and data

Keys, certificates, and data must be implemented as an observable engineering mechanism, not a product label or maturity claim. Map key type, size, purpose, location, owner, lifetime, data, backup, archive, HSM, and replacement path. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the quantum risk, inventory, standardized algorithm, protocol integration, performance, migration, verification, and retirement chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for keys, certificates, and data.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Implementation sketch

```
pqc_migration:
  system: external-api-gateway
  current:
    key_establishment: ecdhe
    signatures: ecdsa
  target:
    key_establishment: hybrid-classical-ml-kem
    signatures: staged-ml-dsa
  gates:
    - official-standard-and-library-support
    - interoperability-matrix-pass
    - mtu-and-fragmentation-pass
    - latency-and-capacity-pass
    - downgrade-protection-pass
    - rollback-tested
  inventory_evidence:
    protocols: [tls, vpn, certificates, hsm, code-signing]
```

Failure patterns

Treating keys, certificates, and data as a product feature instead of an end-to-end engineering system.

Relying on intended policy, rule code, or controller state without proving applied and operational behavior.

Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for keys, certificates, and data.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

2.3 Hardware, firmware, and supplier dependencies

Hardware, firmware, and supplier dependencies must be implemented as an observable engineering mechanism, not a product label or maturity claim. Identify modules, accelerators, smart cards, secure elements, appliances, embedded devices, clouds, and partners. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the quantum risk, inventory, standardized algorithm, protocol integration, performance, migration, verification, and retirement chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric

paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for hardware, firmware, and supplier dependencies.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating hardware, firmware, and supplier dependencies as a product feature instead of an end-to-end engineering system.
- Relying on intended policy, rule code, or controller state without proving applied and operational behavior.
- Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for hardware, firmware, and supplier dependencies.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

2.4 Discovery evidence and confidence

Discovery evidence and confidence must be implemented as an observable engineering mechanism, not a product label or maturity claim. Combine source, binary, configuration, network, certificate, scan, runtime, procurement, and owner evidence. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the quantum risk, inventory, standardized algorithm, protocol integration, performance, migration, verification, and retirement chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for discovery evidence and confidence.

Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.

Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.

Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.

Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.

Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating discovery evidence and confidence as a product feature instead of an end-to-end engineering system.

Relying on intended policy, rule code, or controller state without proving applied and operational behavior.

Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for discovery evidence and confidence.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

Chapter 02 laboratory and review

Practical laboratory
Create a cryptographic inventory for one environment, reconcile four discovery sources, and document blind spots and unsupported assets.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore continuous cryptographic bills of materials and automatically attested algorithm inventories.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 03

NIST PQC standards and algorithm properties

Understand the finalized standards and their distinct operational profiles.

Chapter operating position
Understand the finalized standards and their distinct operational profiles.

3.1 ML-KEM and FIPS 203

ML-KEM and FIPS 203 must be implemented as an observable engineering mechanism, not a product label or maturity claim. Use key generation, encapsulation, decapsulation, parameter sets, shared secrets, input validation, failure behavior, and randomness. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the quantum risk, inventory, standardized algorithm, protocol integration, performance, migration, verification, and retirement chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for ml-kem and fips 203.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating ml-kem and fips 203 as a product feature instead of an end-to-end engineering system.
- Relying on intended policy, rule code, or controller state without proving applied and operational behavior.
- Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for ml-kem and fips 203.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

3.2 ML-DSA and FIPS 204

ML-DSA and FIPS 204 must be implemented as an observable engineering mechanism, not a product label or maturity claim. Use key generation, signing, verification, parameter sets, deterministic or hedged considerations, contexts, and message handling. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the quantum risk, inventory, standardized algorithm, protocol integration, performance, migration, verification, and retirement chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for ml-dsa and fips 204.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Implementation sketch

```
pqc_migration:
  system: external-api-gateway
  current:
    key_establishment: ecdhe
    signatures: ecdsa
  target:
    key_establishment: hybrid-classical-ml-kem
    signatures: staged-ml-dsa
  gates:
    - official-standard-and-library-support
    - interoperability-matrix-pass
    - mtu-and-fragmentation-pass
    - latency-and-capacity-pass
    - downgrade-protection-pass
    - rollback-tested
  inventory_evidence:
    protocols: [tls, vpn, certificates, hsm, code-signing]
```

Failure patterns

Treating ml-dsa and fips 204 as a product feature instead of an end-to-end engineering system.

Relying on intended policy, rule code, or controller state without proving applied and operational behavior.

Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for ml-dsa and fips 204.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

3.3 SLH-DSA and FIPS 205

SLH-DSA and FIPS 205 must be implemented as an observable engineering mechanism, not a product label or maturity claim. Understand stateless hash-based signatures, parameter choices, larger signatures, conservative assumptions, and use cases. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the quantum risk, inventory, standardized algorithm, protocol integration, performance, migration, verification, and retirement chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric

paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for slh-dsa and fips 205.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating slh-dsa and fips 205 as a product feature instead of an end-to-end engineering system.
- Relying on intended policy, rule code, or controller state without proving applied and operational behavior.
- Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for slh-dsa and fips 205.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

3.4 Future and non-final algorithms

Future and non-final algorithms must be implemented as an observable engineering mechanism, not a product label or maturity claim. Track HQC and additional signature candidates without representing selections or drafts as final approved standards. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the quantum risk, inventory, standardized algorithm, protocol integration, performance, migration, verification, and retirement chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for future and non-final algorithms.

Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.

Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.

Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.

Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.

Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating future and non-final algorithms as a product feature instead of an end-to-end engineering system.

Relying on intended policy, rule code, or controller state without proving applied and operational behavior.

Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for future and non-final algorithms.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

Chapter 03 laboratory and review

Practical laboratory
Build an algorithm-selection matrix for key establishment, online signing, firmware signing, offline roots, constrained systems, and long-term archives.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore future HQC standardization, additional signatures, hardware acceleration, and composite cryptographic profiles.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 04

KEM engineering and secure composition

Use key-encapsulation mechanisms as protocol components rather than direct data-encryption algorithms.

Chapter operating position
Use key-encapsulation mechanisms as protocol components rather than direct data-encryption algorithms.

4.1 KEM security model

KEM security model must be implemented as an observable engineering mechanism, not a product label or maturity claim. Understand public and secret keys, encapsulation, ciphertext, shared secret, decapsulation failure, and chosen-ciphertext security. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the quantum risk, inventory, standardized algorithm, protocol integration, performance, migration, verification, and retirement chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for kem security model.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating kem security model as a product feature instead of an end-to-end engineering system.
- Relying on intended policy, rule code, or controller state without proving applied and operational behavior.
- Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for kem security model.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

4.2 KEM-to-channel construction

KEM-to-channel construction must be implemented as an observable engineering mechanism, not a product label or maturity claim. Use KDFs, context, transcript binding, authentication, key confirmation, identities, and downgrade protection. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the quantum risk, inventory, standardized algorithm, protocol integration, performance, migration, verification, and retirement chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for kem-to-channel construction.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Implementation sketch

```
pqc_migration:
  system: external-api-gateway
  current:
    key_establishment: ecdhe
    signatures: ecdsa
  target:
    key_establishment: hybrid-classical-ml-kem
    signatures: staged-ml-dsa
  gates:
    - official-standard-and-library-support
    - interoperability-matrix-pass
    - mtu-and-fragmentation-pass
    - latency-and-capacity-pass
    - downgrade-protection-pass
    - rollback-tested
  inventory_evidence:
    protocols: [tls, vpn, certificates, hsm, code-signing]
```

Failure patterns

Treating kem-to-channel construction as a product feature instead of an end-to-end engineering system.

Relying on intended policy, rule code, or controller state without proving applied and operational behavior.

Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for kem-to-channel construction.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

4.3 Hybrid key establishment

Hybrid key establishment must be implemented as an observable engineering mechanism, not a product label or maturity claim. Combine classical and PQ shared secrets using defined combiners, negotiation, failure behavior, and policy. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the quantum risk, inventory, standardized algorithm, protocol integration, performance, migration, verification, and retirement chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric

paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for hybrid key establishment.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating hybrid key establishment as a product feature instead of an end-to-end engineering system.
- Relying on intended policy, rule code, or controller state without proving applied and operational behavior.
- Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for hybrid key establishment.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

4.4 Implementation and oracle resistance

Implementation and oracle resistance must be implemented as an observable engineering mechanism, not a product label or maturity claim. Use constant-time operations, implicit rejection, validated inputs, randomness, memory safety, and error discipline. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the quantum risk, inventory, standardized algorithm, protocol integration, performance, migration, verification, and retirement chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for implementation and oracle resistance.

Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.

Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.

Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.

Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.

Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating implementation and oracle resistance as a product feature instead of an end-to-end engineering system.

Relying on intended policy, rule code, or controller state without proving applied and operational behavior.

Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for implementation and oracle resistance.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

Chapter 04 laboratory and review

Practical laboratory
Implement or test a KEM-based channel construction using official vectors and negative cases, including malformed ciphertext and context mismatch.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore formally analyzed hybrid combiners, multi-KEM agility, threshold KEMs, and confidential decapsulation.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 05

PQC signatures, PKI, and trust infrastructure

Migrate signing systems while accounting for size, formats, validation, and long-lived trust.

Chapter operating position

Migrate signing systems while accounting for size, formats, validation, and long-lived trust.

5.1 Certificate and public-key representation

Certificate and public-key representation must be implemented as an observable engineering mechanism, not a product label or maturity claim. Plan algorithm identifiers, key formats, extensions, path validation, constraints, trust stores, and interoperability. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the quantum risk, inventory, standardized algorithm, protocol integration, performance, migration, verification, and retirement chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for certificate and public-key representation.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating certificate and public-key representation as a product feature instead of an end-to-end engineering system.
- Relying on intended policy, rule code, or controller state without proving applied and operational behavior.
- Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for certificate and public-key representation.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

5.2 CA and certificate lifecycle

CA and certificate lifecycle must be implemented as an observable engineering mechanism, not a product label or maturity claim. Upgrade roots, intermediates, issuing services, HSMs, enrollment, renewal, revocation, profiles, and relying parties. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the quantum risk, inventory, standardized algorithm, protocol integration, performance, migration, verification, and retirement chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for ca and certificate lifecycle.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Implementation sketch

```
pqc_migration:
  system: external-api-gateway
  current:
    key_establishment: ecdhe
    signatures: ecdsa
  target:
    key_establishment: hybrid-classical-ml-kem
    signatures: staged-ml-dsa
  gates:
    - official-standard-and-library-support
    - interoperability-matrix-pass
    - mtu-and-fragmentation-pass
    - latency-and-capacity-pass
    - downgrade-protection-pass
    - rollback-tested
  inventory_evidence:
    protocols: [tls, vpn, certificates, hsm, code-signing]
```

Failure patterns

- Treating ca and certificate lifecycle as a product feature instead of an end-to-end engineering system.
- Relying on intended policy, rule code, or controller state without proving applied and operational behavior.
- Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for ca and certificate lifecycle.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

5.3 Code, firmware, and artifact signing

Code, firmware, and artifact signing must be implemented as an observable engineering mechanism, not a product label or maturity claim. Handle larger keys and signatures, verification environments, boot chains, update size, rollback, and offline systems. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the quantum risk, inventory, standardized algorithm, protocol integration, performance, migration, verification, and retirement chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric

paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for code, firmware, and artifact signing.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating code, firmware, and artifact signing as a product feature instead of an end-to-end engineering system.
- Relying on intended policy, rule code, or controller state without proving applied and operational behavior.
- Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for code, firmware, and artifact signing.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

5.4 Document, archive, and long-term validation

Document, archive, and long-term validation must be implemented as an observable engineering mechanism, not a product label or maturity claim. Address timestamps, evidence records, re-signing, algorithm expiry, stored verification material, and policy. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the quantum risk, inventory, standardized algorithm, protocol integration, performance, migration, verification, and retirement chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for document, archive, and long-term validation.

Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.

Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.

Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.

Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.

Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating document, archive, and long-term validation as a product feature instead of an end-to-end engineering system.

Relying on intended policy, rule code, or controller state without proving applied and operational behavior.

Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for document, archive, and long-term validation.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

Chapter 05 laboratory and review

Practical laboratory
Design a PQ migration for a code-signing hierarchy and device update chain, including hybrid period, old devices, rollback, and recovery.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore composite certificates, multi-signature trust, post-quantum transparency logs, and long-term evidence renewal.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 06

Protocol migration, performance, and interoperability

Integrate PQC into real systems whose limits were designed around smaller classical artifacts.

Chapter operating position

Integrate PQC into real systems whose limits were designed around smaller classical artifacts.

6.1 TLS, VPN, SSH, and messaging transition

TLS, VPN, SSH, and messaging transition must be implemented as an observable engineering mechanism, not a product label or maturity claim. Assess standards status, hybrid support, negotiation, handshakes, fragmentation, MTU, middleboxes, latency, and fallback. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the quantum risk, inventory, standardized algorithm, protocol integration, performance, migration, verification, and retirement chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for tls, vpn, ssh, and messaging transition.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating tls, vpn, ssh, and messaging transition as a product feature instead of an end-to-end engineering system.
- Relying on intended policy, rule code, or controller state without proving applied and operational behavior.
- Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for tls, vpn, ssh, and messaging transition.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

6.2 Performance and resource analysis

Performance and resource analysis must be implemented as an observable engineering mechanism, not a product label or maturity claim. Measure key generation, encapsulation, decapsulation, signing, verification, bandwidth, memory, CPU, storage, and energy. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the quantum risk, inventory, standardized algorithm, protocol integration, performance, migration, verification, and retirement chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for performance and resource analysis.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Implementation sketch

```
pqc_migration:
  system: external-api-gateway
  current:
    key_establishment: ecdhe
    signatures: ecdsa
  target:
    key_establishment: hybrid-classical-ml-kem
    signatures: staged-ml-dsa
  gates:
    - official-standard-and-library-support
    - interoperability-matrix-pass
    - mtu-and-fragmentation-pass
    - latency-and-capacity-pass
    - downgrade-protection-pass
    - rollback-tested
  inventory_evidence:
    protocols: [tls, vpn, certificates, hsm, code-signing]
```

Failure patterns

- Treating performance and resource analysis as a product feature instead of an end-to-end engineering system.
- Relying on intended policy, rule code, or controller state without proving applied and operational behavior.
- Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for performance and resource analysis.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

6.3 Constrained and embedded systems

Constrained and embedded systems must be implemented as an observable engineering mechanism, not a product label or maturity claim. Account for firmware, RAM, flash, radio frames, boot time, power, update paths, hardware support, and service life. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the quantum risk, inventory, standardized algorithm, protocol integration, performance, migration, verification, and retirement chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric

paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for constrained and embedded systems.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating constrained and embedded systems as a product feature instead of an end-to-end engineering system.
- Relying on intended policy, rule code, or controller state without proving applied and operational behavior.
- Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for constrained and embedded systems.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

6.4 Interoperability and downgrade testing

Interoperability and downgrade testing must be implemented as an observable engineering mechanism, not a product label or maturity claim. Test peers, versions, providers, libraries, algorithms, failures, fallback, policy, and telemetry across vendors. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the quantum risk, inventory, standardized algorithm, protocol integration, performance, migration, verification, and retirement chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for interoperability and downgrade testing.

Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.

Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.

Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.

Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.

Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating interoperability and downgrade testing as a product feature instead of an end-to-end engineering system.

Relying on intended policy, rule code, or controller state without proving applied and operational behavior.

Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for interoperability and downgrade testing.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

Chapter 06 laboratory and review

Practical laboratory
Create a protocol and platform benchmark comparing classical, PQ, and hybrid modes under normal, lossy, high-latency, and constrained conditions.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore PQC for 5G/6G, satellite links, optical control, industrial systems, and agentic service meshes.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 07

Enterprise migration program and governance

Coordinate a multi-year transition across technology, suppliers, risk, and operations.

Chapter operating position
Coordinate a multi-year transition across technology, suppliers, risk, and operations.

7.1 Target architecture and policy

Target architecture and policy must be implemented as an observable engineering mechanism, not a product label or maturity claim. Define approved algorithms, hybrid use, exceptions, data and trust lifetimes, validation, vendors, and retirement. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the quantum risk, inventory, standardized algorithm, protocol integration, performance, migration, verification, and retirement chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for target architecture and policy.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating target architecture and policy as a product feature instead of an end-to-end engineering system.
- Relying on intended policy, rule code, or controller state without proving applied and operational behavior.
- Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for target architecture and policy.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

7.2 Phases and prioritization

Phases and prioritization must be implemented as an observable engineering mechanism, not a product label or maturity claim. Sequence discovery, pilots, foundational services, high-value data, new systems, legacy remediation, and deprecation. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the quantum risk, inventory, standardized algorithm, protocol integration, performance, migration, verification, and retirement chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for phases and prioritization.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Implementation sketch

```
pqc_migration:
  system: external-api-gateway
  current:
    key_establishment: ecdhe
    signatures: ecdsa
  target:
    key_establishment: hybrid-classical-ml-kem
    signatures: staged-ml-dsa
  gates:
    - official-standard-and-library-support
    - interoperability-matrix-pass
    - mtu-and-fragmentation-pass
    - latency-and-capacity-pass
    - downgrade-protection-pass
    - rollback-tested
  inventory_evidence:
    protocols: [tls, vpn, certificates, hsm, code-signing]
```

Failure patterns

Treating phases and prioritization as a product feature instead of an end-to-end engineering system.

Relying on intended policy, rule code, or controller state without proving applied and operational behavior.

Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for phases and prioritization.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

7.3 Supplier and acquisition requirements

Supplier and acquisition requirements must be implemented as an observable engineering mechanism, not a product label or maturity claim. Require inventory, roadmaps, standards status, interoperability, updates, evidence, support, and exit plans. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the quantum risk, inventory, standardized algorithm, protocol integration, performance, migration, verification, and retirement chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric

paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for supplier and acquisition requirements.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating supplier and acquisition requirements as a product feature instead of an end-to-end engineering system.
- Relying on intended policy, rule code, or controller state without proving applied and operational behavior.
- Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for supplier and acquisition requirements.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

7.4 Change, incident, and rollback

Change, incident, and rollback must be implemented as an observable engineering mechanism, not a product label or maturity claim. Prepare for implementation defects, algorithm findings, interoperability failures, performance regressions, and emergency policy. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the quantum risk, inventory, standardized algorithm, protocol integration, performance, migration, verification, and retirement chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for change, incident, and rollback.

Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.

Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.

Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.

Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.

Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating change, incident, and rollback as a product feature instead of an end-to-end engineering system.

Relying on intended policy, rule code, or controller state without proving applied and operational behavior.

Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for change, incident, and rollback.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

Chapter 07 laboratory and review

Practical laboratory
Build a five-wave enterprise PQC roadmap with dependencies, owners, gates, suppliers, metrics, and explicit classical retirement criteria.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Develop a governed crypto-migration control plane that tracks every system, algorithm, deadline, exception, and proof.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 08

Testing, validation, operations, and future readiness

Prove the migration works and remains adaptable as standards and implementations evolve.

Chapter operating position

Prove the migration works and remains adaptable as standards and implementations evolve.

8.1 Known-answer and conformance testing

Known-answer and conformance testing must be implemented as an observable engineering mechanism, not a product label or maturity claim. Use official vectors, deterministic fixtures, negative inputs, error cases, parameter validation, and cross-implementation comparison. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the quantum risk, inventory, standardized algorithm, protocol integration, performance, migration, verification, and retirement chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for known-answer and conformance testing.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating known-answer and conformance testing as a product feature instead of an end-to-end engineering system.
- Relying on intended policy, rule code, or controller state without proving applied and operational behavior.
- Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for known-answer and conformance testing.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

8.2 Security and implementation testing

Security and implementation testing must be implemented as an observable engineering mechanism, not a product label or maturity claim. Assess side channels, fault behavior, randomness, memory safety, parsing, key storage, error oracles, and supply chain. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the quantum risk, inventory, standardized algorithm, protocol integration, performance, migration, verification, and retirement chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for security and implementation testing.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Implementation sketch

```
pqc_migration:
  system: external-api-gateway
  current:
    key_establishment: ecdhe
    signatures: ecdsa
  target:
    key_establishment: hybrid-classical-ml-kem
    signatures: staged-ml-dsa
  gates:
    - official-standard-and-library-support
    - interoperability-matrix-pass
    - mtu-and-fragmentation-pass
    - latency-and-capacity-pass
    - downgrade-protection-pass
    - rollback-tested
  inventory_evidence:
    protocols: [tls, vpn, certificates, hsm, code-signing]
```

Failure patterns

- Treating security and implementation testing as a product feature instead of an end-to-end engineering system.
- Relying on intended policy, rule code, or controller state without proving applied and operational behavior.
- Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for security and implementation testing.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

8.3 Operational telemetry and inventory drift

Operational telemetry and inventory drift must be implemented as an observable engineering mechanism, not a product label or maturity claim. Monitor algorithms, handshakes, certificates, failures, fallback, performance, modules, policy, and unauthorized classical use. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the quantum risk, inventory, standardized algorithm, protocol integration, performance, migration, verification, and retirement chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric

paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

- Define authority, ownership, scope, dependencies, limits, and acceptance criteria for operational telemetry and inventory drift.
- Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.
- Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.
- Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.
- Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.
- Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

- Treating operational telemetry and inventory drift as a product feature instead of an end-to-end engineering system.
- Relying on intended policy, rule code, or controller state without proving applied and operational behavior.
- Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, or partial failure.
- Changing several variables before preserving evidence and stating a falsifiable hypothesis.
- Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.
- Declaring success after one positive result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for operational telemetry and inventory drift.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

8.4 Freshness and standards tracking

Freshness and standards tracking must be implemented as an observable engineering mechanism, not a product label or maturity claim. Track FIPS updates, SP 800-227, draft transition guidance, HQC work, protocol standards, modules, and verified errata. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the quantum risk, inventory, standardized algorithm, protocol integration, performance, migration, verification, and retirement chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

A production implementation starts with a minimal known-good baseline and explicit invariants. Introduce one dependency or capability at a time, preserve before-state evidence, test authorized and prohibited behavior, inject realistic failures, measure convergence or recovery, and retain a tested rollback. Automation must stop safely when authority, freshness, identity, integrity, cryptographic state, or observed outcomes are ambiguous.

Troubleshooting locates the first divergence from the expected state machine or end-to-end transaction. Inspect every authority boundary and account for time, caching, eventual consistency, stale identity, incomplete collection, parser drift, asymmetric paths, encapsulation, supply-chain state, side channels, partial failure, and missing evidence. Closure requires causal explanation, reproducibility, validated restoration, residual limitations, and prevention or monitoring actions.

Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for freshness and standards tracking.

Model expected state, trust boundaries, data flow, identity, and failure behavior before implementation.

Validate intended configuration, active state, applied enforcement, raw evidence, and end-to-end outcome independently.

Test normal, prohibited, degraded, failed, scale, recovery, and rollback conditions.

Preserve policy, source, version, identity, telemetry, artifact, packet, log, time, and decision evidence.

Automate through typed contracts, least privilege, deterministic gates, canaries, bounded authority, and reversible changes.

Failure patterns

Treating freshness and standards tracking as a product feature instead of an end-to-end engineering system.

Relying on intended policy, rule code, or controller state without proving applied and operational behavior.

Ignoring data quality, identity, time, cryptographic lifecycle, supplier state, or partial failure.

Changing several variables before preserving evidence and stating a falsifiable hypothesis.

Allowing automation or AI to act on stale, conflicting, unverified, or unauthorized inputs.

Declaring success after one positive result without negative tests, rollback, and recurrence monitoring.

Validation and evidence

Method	Expected evidence
Examine	Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for freshness and standards tracking.
Observe	Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior.
Test	Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback.
Measure	Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome.
Reconcile	Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records.

Chapter 08 laboratory and review

Practical laboratory
Create a PQC acceptance suite and continuous monitoring plan for one protocol, PKI, and signing workflow.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore self-describing cryptographic systems, automated safe algorithm transition, and quantum-network coexistence.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

Integrated learning roadmap

The guide is intended to be revisited. First establish fluency, then build a small implementation, operate it under failure, automate repeatable work, and finally evaluate emerging techniques against a stable baseline. Progress is demonstrated through evidence rather than reading completion alone.

Stage	Demonstrated capability
Foundation	Explain the system from first principles and trace its critical path without relying on product terminology.
Build	Implement the smallest representative system with explicit contracts, identities, telemetry, and tests.
Operate	Diagnose injected failures, recover safely, and preserve evidence of the causal chain.
Automate	Convert a proven manual workflow into bounded, idempotent, observable automation.
Architect	Design for scale, resilience, security, interoperability, and lifecycle change.
Explore	Evaluate frontier techniques using stated hypotheses, limitations, and adoption gates.

Source authority register

Source policy
Official standards, specifications, and project documentation remain with their issuing organizations. Mythos Systems provides original engineering interpretation, synthesis, implementation guidance, and relationship mapping. Source verification dates do not guarantee that an authority has not changed since review.

01. NIST - Post-Quantum Cryptography Project
Role: Official PQC standardization, migration, publications, and status source.
Verified: 2026-07-26
Official source: https://csrc.nist.gov/projects/post-quantum-cryptography

02. NIST - FIPS 203 - Module-Lattice-Based Key-Encapsulation Mechanism Standard
Role: ML-KEM standard.
Verified: 2026-07-26
Official source: https://csrc.nist.gov/pubs/fips/203/final

03. NIST - FIPS 204 - Module-Lattice-Based Digital Signature Standard
Role: ML-DSA standard.
Verified: 2026-07-26
Official source: https://csrc.nist.gov/pubs/fips/204/final

04. NIST - FIPS 205 - Stateless Hash-Based Digital Signature Standard
Role: SLH-DSA standard.
Verified: 2026-07-26
Official source: https://csrc.nist.gov/pubs/fips/205/final

05. NIST - SP 800-227 - Recommendations for Key-Encapsulation Mechanisms
Role: Final KEM definitions, properties, applications, and secure-use guidance.
Verified: 2026-07-26
Official source: https://csrc.nist.gov/pubs/sp/800/227/final

06. NIST - Draft NIST IR 8547 - Transition to Post-Quantum Cryptography Standards
Role: Draft transition roadmap; explicitly treated as non-final.
Verified: 2026-07-26
Official source: https://csrc.nist.gov/pubs/ir/8547/ipd

07. NIST NCCoE - Migration to Post-Quantum Cryptography
Role: Cryptographic discovery, interoperability, and migration project resources.
Verified: 2026-07-26
Official source: https://pages.nist.gov/nccoe-migration-post-quantum-cryptography/

08. NIST - HQC Fourth-Round Selection Announcement
Role: Official status of HQC as a selected future KEM, not yet a final standard.
Verified: 2026-07-26
Official source: https://csrc.nist.gov/news/2025/hqc-announced-as-a-4th-round-selection

09. NIST - Draft SP 800-57 Part 1 Rev. 6 - Recommendation for Key Management
Role: Draft modernization including quantum-resistant algorithms; explicitly treated as draft.
Verified: 2026-07-26
Official source: https://csrc.nist.gov/pubs/sp/800/57/pt1/r6/ipd

Authority application and refresh triggers

Authority class	Required library action
Protocols and specifications	Recheck interoperability, security considerations, defaults, and migration guidance when a referenced specification changes.
Security and assurance frameworks	Update control mappings, evidence expectations, risk language, and assessment procedures after material framework revision.
Languages, runtimes, and projects	Re-run examples, tests, deprecation checks, and operational recommendations against supported releases.
Benchmarks and evaluations	Re-execute claims when workloads, methods, models, compilers, drivers, hardware, or scoring rules change.
Operational evidence	Incorporate validated incidents, failure tests, reader corrections, and measured implementation outcomes into the next revision.

Limitations, freshness, and revision control

Boundary	Statement
Publication limitation	This guide synthesizes broad engineering practice. It does not replace product documentation, legal advice, safety certification, or environment-specific design review.
Evidence limitation	Not every pattern is benchmarked in every environment. Examples illustrate engineering methods and must be validated against actual versions, workloads, hardware, policy, and failure conditions.
Freshness limitation	Vendor behavior, software libraries, AI models, security threats, and emerging standards can change quickly. Volatile claims require source revalidation before operational adoption.
Adoption limitation	Frontier content is not an implicit adoption recommendation. Adoption requires defined value, qualification gates, controlled proof, rollback, ownership, and residual-risk acceptance.
Status boundary	Candidate Focused Field Guide describes the publication, not certification of a product, implementation, engineer, or organization.

Revision record

Version	Revision statement
1.0.0-candidate	Initial candidate living-guide edition. Published 2026-07-26; scheduled review 2026-10-26.

Search and relationship metadata

Dimension	Engineering position
Primary domain	MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography
Secondary domain	MYTHOS-QTC; MYTHOS-NET; MYTHOS-SWE
Publication class	Field Guide / Canonical Living Overview Guide
Skill levels	L1 Foundational; L2 Practitioner; L3 Advanced; L4 Frontier
Tags	post-quantum cryptography, PQC, ML-KEM, ML-DSA, SLH-DSA, FIPS 203, FIPS 204, FIPS 205, crypto migration, hybrid cryptography
Canonical route	/library/field-guides/mythos-fg-sec-0017/

Living publication

Corrections, expanded laboratories, improved diagrams, authority changes, and new engineering evidence should revise this permanent publication rather than create disconnected replacements.