CANONICAL LIVING OVERVIEW GUIDE

Modern Cryptography and Post-Quantum Migration

A practical cryptographic engineering guide from primitives, protocols, PKI, HSMs, and key lifecycle through crypto-agility, NIST post-quantum standards, hybrid transition, protocol migration, inventory, testing, and long-lived data risk.

PERMANENT ID
MYTHOS-GUIDE-PQC-0001

PUBLICATION CLASS
FIELD GUIDE

VERSION / STATUS
1.0.0-candidate / CANDIDATE

PERMANENT PUBLICATION IDENTITY

Modern Cryptography and Post-Quantum Migration

Document ID
MYTHOS-GUIDE-PQC-0001

Version
1.0.0-candidate

Status
Candidate Living 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; MYTHOS-CLD
Audience	Security architects, cryptographic engineers, PKI teams, network and software engineers, infrastructure owners, risk leaders, and technical decision makers.
Prerequisites	Security fundamentals, networking, software architecture, and basic public-key cryptography.
Publisher / owner	Mythos Systems / Aaron Stovall
Public classification	Public technical guidance

Expected outcomes

Explain cryptographic goals, primitives, protocols, implementation hazards, and trust dependencies.

Inventory cryptographic usage across code, protocols, hardware, services, data, and supply chains.

Plan and execute crypto-agile transition to ML-KEM, ML-DSA, and SLH-DSA with hybrid interoperability.

Evaluate performance, size, certificate, HSM, network, application, and operational impacts.

Separate post-quantum cryptography from QKD and from unsupported quantum-security claims.

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

Cryptographic engineering foundations

Connect security goals to primitives, protocols, implementations, identities, keys, and operational trust.

Chapter operating position
Connect security goals to primitives, protocols, implementations, identities, keys, and operational trust.

1.1 Confidentiality, integrity, authenticity, and non-repudiation

Confidentiality, integrity, authenticity, and non-repudiation is treated here as an engineering capability rather than a vocabulary item. Define the actual property required, adversary, lifetime, context, and acceptable failure. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for confidentiality, integrity, authenticity, and non-repudiation.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating confidentiality, integrity, authenticity, and non-repudiation as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for confidentiality, integrity, authenticity, and non-repudiation.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

1.2 Symmetric encryption, hashes, MACs, and KDFs

Symmetric encryption, hashes, MACs, and KDFs is treated here as an engineering capability rather than a vocabulary item. Use modes, nonces, tags, salts, domains, derivation, and parameter choices correctly. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for symmetric encryption, hashes, macs, and kdfs.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Implementation sketch

```
use std::sync::Arc;
use tokio::sync::Semaphore;

pub async fn execute_symmetric_encryption_hashes_macs_and_kdfs(
    limit: Arc<Semaphore>,
) -> anyhow::Result<()> {
    let _permit = limit.acquire().await?;
    // Validate inputs, preserve cancellation, and emit structured evidence.
    Ok(())
}
```

Failure patterns

- Treating symmetric encryption, hashes, macs, and kdfs as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for symmetric encryption, hashes, macs, and kdfs.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

1.3 Public-key encryption, signatures, and key agreement

Public-key encryption, signatures, and key agreement is treated here as an engineering capability rather than a vocabulary item. Distinguish encryption, encapsulation, signing, authentication, and agreement and their security assumptions. The design

objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for public-key encryption, signatures, and key agreement.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating public-key encryption, signatures, and key agreement as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for public-key encryption, signatures, and key agreement.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

1.4 Randomness, entropy, and implementation security

Randomness, entropy, and implementation security is treated here as an engineering capability rather than a vocabulary item. Engineer entropy sources, DRBGs, side-channel resistance, constant-time code, fault handling, and testing. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency,

orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for randomness, entropy, and implementation security.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating randomness, entropy, and implementation security as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for randomness, entropy, and implementation security.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

Chapter 01 laboratory and review

Practical laboratory
Create a cryptographic data-flow diagram for an application and identify every algorithm, key, trust anchor, random source, protocol, and data lifetime.

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

Compare classical, PQC, QKD, threshold, and confidential-computing approaches by property rather than by label.

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

Protocols, PKI, identities, and trust

Build complete trust systems in which algorithms are only one component.

Chapter operating position
Build complete trust systems in which algorithms are only one component.

2.1 Certificates, trust anchors, and path validation

Certificates, trust anchors, and path validation is treated here as an engineering capability rather than a vocabulary item. Engineer names, key usage, constraints, revocation, trust stores, issuance, and validation. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for certificates, trust anchors, and path validation.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating certificates, trust anchors, and path validation as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for certificates, trust anchors, and path validation.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

2.2 TLS, SSH, IPsec, S/MIME, code signing, and application crypto

TLS, SSH, IPsec, S/MIME, code signing, and application crypto is treated here as an engineering capability rather than a vocabulary item. Map cryptographic functions and dependencies across common protocols and artifact lifecycles. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for tls, ssh, ipsec, s/mime, code signing, and application crypto.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Implementation sketch

```
use std::sync::Arc;
use tokio::sync::Semaphore;

pub async fn execute_tls_ssh_ipsec_s_mime_code_signing_and_application_crypto(
    limit: Arc<Semaphore>,
) -> anyhow::Result<()> {
    let _permit = limit.acquire().await?;
    // Validate inputs, preserve cancellation, and emit structured evidence.
    Ok(())
}
```

Failure patterns

- Treating tls, ssh, ipsec, s/mime, code signing, and application crypto as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for tls, ssh, ipsec, s/mime, code signing, and application crypto.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

2.3 Key management, rotation, escrow, and recovery

Key management, rotation, escrow, and recovery is treated here as an engineering capability rather than a vocabulary item. Define generation, custody, activation, use, backup, rotation, compromise, destruction, and evidence. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for key management, rotation, escrow, and recovery.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating key management, rotation, escrow, and recovery as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for key management, rotation, escrow, and recovery.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

2.4 HSMs, TPMs, secure elements, and remote services

HSMs, TPMs, secure elements, and remote services is treated here as an engineering capability rather than a vocabulary item. Evaluate hardware boundaries, APIs, throughput, availability, attestation, backup, and vendor dependency. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency,

orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for hsms, tpms, secure elements, and remote services.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating hsms, tpms, secure elements, and remote services as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for hsms, tpms, secure elements, and remote services.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

Chapter 02 laboratory and review

Practical laboratory

Trace certificate and key lifecycle for a service from enrollment through TLS use, rotation, incident revocation, 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

Assess decentralized identifiers, threshold PKI, hardware-rooted agent identities, and post-quantum certificate models.

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

Quantum threat and migration rationale

Understand which cryptographic assumptions are threatened, which remain viable, and why migration must begin before a cryptographically relevant quantum computer exists.

Chapter operating position

Understand which cryptographic assumptions are threatened, which remain viable, and why migration must begin before a cryptographically relevant quantum computer exists.

3.1 Shor, Grover, and affected algorithm classes

Shor, Grover, and affected algorithm classes is treated here as an engineering capability rather than a vocabulary item. Relate quantum algorithms to RSA, finite-field and elliptic-curve systems, symmetric keys, and hashes. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for shor, grover, and affected algorithm classes.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating shor, grover, and affected algorithm classes as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for shor, grover, and affected algorithm classes.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

3.2 Harvest-now-decrypt-later risk

Harvest-now-decrypt-later risk is treated here as an engineering capability rather than a vocabulary item. Prioritize data whose confidentiality lifetime exceeds the expected migration and adversary collection window. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for harvest-now-decrypt-later risk.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Implementation sketch

```
use std::sync::Arc;
use tokio::sync::Semaphore;

pub async fn execute_harvest_now_decrypt_later_risk(
    limit: Arc<Semaphore>,
) -> anyhow::Result<()> {
    let _permit = limit.acquire().await?;
    // Validate inputs, preserve cancellation, and emit structured evidence.
    Ok(())
}
```

Failure patterns

- Treating harvest-now-decrypt-later risk as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for harvest-now-decrypt-later risk.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

3.3 Migration timelines and dependency depth

Migration timelines and dependency depth is treated here as an engineering capability rather than a vocabulary item. Account for inventories, standards, implementations, protocols, hardware, supply chains, certification, and long replacement cycles. The

design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for migration timelines and dependency depth.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating migration timelines and dependency depth as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for migration timelines and dependency depth.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

3.4 PQC, QKD, and quantum-safe distinctions

PQC, QKD, and quantum-safe distinctions is treated here as an engineering capability rather than a vocabulary item. Separate mathematical post-quantum algorithms, physics-based key distribution, and broader quantum-readiness programs. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

Define the authoritative source and ownership boundary for pqc, qkd, and quantum-safe distinctions.

Make preconditions, invariants, error states, and recovery actions explicit.

Instrument the critical path with stable identifiers, timestamps, and outcome codes.

Validate in a representative environment before widening blast radius.

Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

Treating pqc, qkd, and quantum-safe distinctions as a product feature rather than an end-to-end engineering behavior.

Automating an undocumented process and scaling its ambiguity.

Relying on a happy-path demonstration without degraded-state or rollback testing.

Using aggregate dashboards without retaining trace-level evidence.

Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for pqc, qkd, and quantum-safe distinctions.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

Chapter 03 laboratory and review

Practical laboratory
Classify organizational data by confidentiality lifetime, exposure, current protection, migration complexity, and harvest-now risk.

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

Model uncertainty in quantum timelines without using a single predicted date as the migration decision.

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

NIST post-quantum standards

Engineer with finalized standards while preserving algorithm and implementation agility.

Chapter operating position
Engineer with finalized standards while preserving algorithm and implementation agility.

4.1 ML-KEM and key establishment

ML-KEM and key establishment is treated here as an engineering capability rather than a vocabulary item. Understand parameter sets, key and ciphertext sizes, encapsulation, decapsulation, failure behavior, and integration. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for ml-kem and key establishment.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating ml-kem and key establishment as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for ml-kem and key establishment.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

4.2 ML-DSA and digital signatures

ML-DSA and digital signatures is treated here as an engineering capability rather than a vocabulary item. Understand parameter sets, signing, verification, key and signature size, randomness, and operational use. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for ml-dsa and digital signatures.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Implementation sketch

```
use std::sync::Arc;
use tokio::sync::Semaphore;

pub async fn execute_ml_dsa_and_digital_signatures(
    limit: Arc<Semaphore>,
) -> anyhow::Result<()> {
    let _permit = limit.acquire().await?;
    // Validate inputs, preserve cancellation, and emit structured evidence.
    Ok(())
}
```

Failure patterns

- Treating ml-dsa and digital signatures as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for ml-dsa and digital signatures.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

4.3 SLH-DSA and hash-based signatures

SLH-DSA and hash-based signatures is treated here as an engineering capability rather than a vocabulary item. Understand stateless hash-based design, larger signatures, performance, conservative assumptions, and use cases. The design objective is

to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for slh-dsa and hash-based signatures.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating slh-dsa and hash-based signatures as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for slh-dsa and hash-based signatures.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

4.4 Additional algorithms and standards lifecycle

Additional algorithms and standards lifecycle is treated here as an engineering capability rather than a vocabulary item. Track FIPS 206, backup KEMs, parameter revisions, implementation guidance, validation, and cryptanalysis. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

Define the authoritative source and ownership boundary for additional algorithms and standards lifecycle.

Make preconditions, invariants, error states, and recovery actions explicit.

Instrument the critical path with stable identifiers, timestamps, and outcome codes.

Validate in a representative environment before widening blast radius.

Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

Treating additional algorithms and standards lifecycle as a product feature rather than an end-to-end engineering behavior.

Automating an undocumented process and scaling its ambiguity.

Relying on a happy-path demonstration without degraded-state or rollback testing.

Using aggregate dashboards without retaining trace-level evidence.

Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for additional algorithms and standards lifecycle.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

Chapter 04 laboratory and review

Practical laboratory
Benchmark supported ML-KEM, ML-DSA, and SLH-DSA implementations for key generation, operation latency, size, memory, concurrency, and failure handling.

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

Track additional signature standards, alternative KEMs, compact signatures, and hardware acceleration as evolving work.

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

Crypto inventory and agility architecture

Make cryptographic dependencies discoverable, replaceable, testable, and governed.

Chapter operating position
Make cryptographic dependencies discoverable, replaceable, testable, and governed.

5.1 Cryptographic bill of materials

Cryptographic bill of materials is treated here as an engineering capability rather than a vocabulary item. Inventory algorithms, parameters, protocols, certificates, libraries, devices, services, data, and ownership. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for cryptographic bill of materials.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating cryptographic bill of materials as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for cryptographic bill of materials.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

5.2 Abstraction and policy boundaries

Abstraction and policy boundaries is treated here as an engineering capability rather than a vocabulary item. Separate cryptographic policy from implementation and avoid hard-coded algorithm, provider, key, and certificate assumptions. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for abstraction and policy boundaries.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Implementation sketch

```
use std::sync::Arc;
use tokio::sync::Semaphore;

pub async fn execute_abstraction_and_policy_boundaries(
    limit: Arc<Semaphore>,
) -> anyhow::Result<()> {
    let _permit = limit.acquire().await?;
    // Validate inputs, preserve cancellation, and emit structured evidence.
    Ok(())
}
```

Failure patterns

- Treating abstraction and policy boundaries as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for abstraction and policy boundaries.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

5.3 Negotiation, identifiers, and downgrade protection

Negotiation, identifiers, and downgrade protection is treated here as an engineering capability rather than a vocabulary item. Version algorithms and parameter suites, preserve interoperability, prevent downgrade, and log negotiated outcomes. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for negotiation, identifiers, and downgrade protection.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating negotiation, identifiers, and downgrade protection as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for negotiation, identifiers, and downgrade protection.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

5.4 Agility tests and emergency replacement

Agility tests and emergency replacement is treated here as an engineering capability rather than a vocabulary item. Exercise provider changes, algorithm disablement, key replacement, certificate reissue, rollback, and incident response. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for agility tests and emergency replacement.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating agility tests and emergency replacement as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for agility tests and emergency replacement.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

Chapter 05 laboratory and review

Practical laboratory
Build a crypto inventory for one service and replace its cryptographic provider or algorithm suite without changing application business logic.

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 machine-readable crypto policy, continuous discovery, and automated migration evidence.

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

Hybrid migration and protocol engineering

Combine classical and post-quantum mechanisms during transition while recognizing size, latency, interoperability, and draft-status constraints.

Chapter operating position

Combine classical and post-quantum mechanisms during transition while recognizing size, latency, interoperability, and draft-status constraints.

6.1 Hybrid key agreement principles

Hybrid key agreement principles is treated here as an engineering capability rather than a vocabulary item. Combine independent secrets safely, negotiate as a single method, preserve transcript binding, and define failure. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for hybrid key agreement principles.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating hybrid key agreement principles as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for hybrid key agreement principles.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

6.2 TLS migration

TLS migration is treated here as an engineering capability rather than a vocabulary item. Evaluate supported groups, certificates, handshakes, fragmentation, middleboxes, load balancers, inspection, and monitoring. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for tls migration.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Implementation sketch

```
use std::sync::Arc;
use tokio::sync::Semaphore;

pub async fn execute_tls_migration(
    limit: Arc<Semaphore>,
) -> anyhow::Result<()> {
    let _permit = limit.acquire().await?;
    // Validate inputs, preserve cancellation, and emit structured evidence.
    Ok(())
}
```

Failure patterns

- Treating tls migration as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for tls migration.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

6.3 SSH, IPsec, EAP, VPN, and remote-access migration

SSH, IPsec, EAP, VPN, and remote-access migration is treated here as an engineering capability rather than a vocabulary item. Track protocol specifications, vendor support, packet size, negotiation, gateways, identity, and fallback. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for ssh, ipsec, eap, vpn, and remote-access migration.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating ssh, ipsec, eap, vpn, and remote-access migration as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for ssh, ipsec, eap, vpn, and remote-access migration.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

6.4 PKI and signature migration

PKI and signature migration is treated here as an engineering capability rather than a vocabulary item. Plan certificate profiles, roots, intermediates, code signing, firmware, timestamping, validation, and long-lived artifacts. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for pki and signature migration.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating pki and signature migration as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for pki and signature migration.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

Chapter 06 laboratory and review

Practical laboratory
Create a hybrid TLS test matrix across client, server, proxy, load balancer, inspection device, network path, and certificate configurations.

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

Evaluate pure-PQC transition criteria, composite certificates, and post-quantum network access as specifications mature.

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

Implementation, performance, and operational assurance

Deploy PQC without turning new algorithms into new availability, side-channel, or lifecycle failures.

Chapter operating position
Deploy PQC without turning new algorithms into new availability, side-channel, or lifecycle failures.

7.1 Library and provider selection

Library and provider selection is treated here as an engineering capability rather than a vocabulary item. Evaluate standards conformance, validation, constant-time behavior, maintenance, language bindings, platform support, and provenance. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for library and provider selection.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating library and provider selection as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for library and provider selection.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

7.2 Performance, size, fragmentation, and capacity

Performance, size, fragmentation, and capacity is treated here as an engineering capability rather than a vocabulary item. Measure CPU, memory, keys, signatures, ciphertexts, packets, handshakes, storage, queues, and device limits. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for performance, size, fragmentation, and capacity.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Implementation sketch

```
use std::sync::Arc;
use tokio::sync::Semaphore;

pub async fn execute_performance_size_fragmentation_and_capacity(
    limit: Arc<Semaphore>,
) -> anyhow::Result<()> {
    let _permit = limit.acquire().await?;
    // Validate inputs, preserve cancellation, and emit structured evidence.
    Ok(())
}
```

Failure patterns

- Treating performance, size, fragmentation, and capacity as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for performance, size, fragmentation, and capacity.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

7.3 Hardware, HSM, accelerator, and embedded support

Hardware, HSM, accelerator, and embedded support is treated here as an engineering capability rather than a vocabulary item. Assess firmware, APIs, key storage, throughput, certification, upgrade, and replacement timelines. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for hardware, hsm, accelerator, and embedded support.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating hardware, hsm, accelerator, and embedded support as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for hardware, hsm, accelerator, and embedded support.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

7.4 Testing, monitoring, and incident response

Testing, monitoring, and incident response is treated here as an engineering capability rather than a vocabulary item. Use known-answer, interoperability, negative, fault, side-channel, negotiation, expiry, and recovery tests with telemetry. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency,

orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for testing, monitoring, and incident response.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating testing, monitoring, and incident response as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for testing, monitoring, and incident response.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

Chapter 07 laboratory and review

Practical laboratory
Deploy PQC in a controlled service and test load, packet fragmentation, certificate chains, failures, observability, rotation, and rollback.

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 optical cryptography, confidential acceleration, threshold PQC, and formal verification of implementations.

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

Enterprise migration program and future posture

Coordinate technical migration across business priorities, procurement, suppliers, architecture, operations, and evidence.

Chapter operating position
Coordinate technical migration across business priorities, procurement, suppliers, architecture, operations, and evidence.

8.1 Discovery and prioritization

Discovery and prioritization is treated here as an engineering capability rather than a vocabulary item. Identify exposure, data lifetime, critical services, dependencies, owners, replacement difficulty, and migration sequence. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for discovery and prioritization.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating discovery and prioritization as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for discovery and prioritization.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

8.2 Roadmaps, pilots, and coexistence

Roadmaps, pilots, and coexistence is treated here as an engineering capability rather than a vocabulary item. Define stages for inventory, laboratory, hybrid pilot, limited production, broad migration, and classical retirement. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for roadmaps, pilots, and coexistence.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Implementation sketch

```
use std::sync::Arc;
use tokio::sync::Semaphore;

pub async fn execute_roadmaps_pilots_and_coexistence(
    limit: Arc<Semaphore>,
) -> anyhow::Result<()> {
    let _permit = limit.acquire().await?;
    // Validate inputs, preserve cancellation, and emit structured evidence.
    Ok(())
}
```

Failure patterns

- Treating roadmaps, pilots, and coexistence as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for roadmaps, pilots, and coexistence.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

8.3 Supplier and product readiness

Supplier and product readiness is treated here as an engineering capability rather than a vocabulary item. Require algorithm support, roadmaps, SBOM/CBOM, APIs, validation, testing, firmware, lifecycle, and disclosure. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for supplier and product readiness.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating supplier and product readiness as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for supplier and product readiness.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

8.4 Governance, evidence, and revalidation

Governance, evidence, and revalidation is treated here as an engineering capability rather than a vocabulary item. Track decisions, exceptions, residual risk, incidents, standards changes, cryptanalysis, and completion proof. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

Define the authoritative source and ownership boundary for governance, evidence, and revalidation.

Make preconditions, invariants, error states, and recovery actions explicit.

Instrument the critical path with stable identifiers, timestamps, and outcome codes.

Validate in a representative environment before widening blast radius.

Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

Treating governance, evidence, and revalidation as a product feature rather than an end-to-end engineering behavior.

Automating an undocumented process and scaling its ambiguity.

Relying on a happy-path demonstration without degraded-state or rollback testing.

Using aggregate dashboards without retaining trace-level evidence.

Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for governance, evidence, and revalidation.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

Chapter 08 laboratory and review

Practical laboratory

Produce a five-year migration roadmap for a mixed enterprise containing web PKI, VPN, SSH, network access, code signing, HSMs, embedded devices, and long-lived 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

Define decision gates for QKD, quantum networking, and future cryptographic techniques without allowing them to delay deployable PQC.

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

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

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

04. NIST - Post-Quantum Cryptography Project
Role: PQC standards, status, migration, and additional algorithm work.
Verified: 2026-07-26
Official source: https://csrc.nist.gov/projects/post-quantum-cryptography

05. NIST - Crypto Agility Project and CSWP 39
Role: Cryptographic inventory, replacement, interoperability, and operational agility guidance.
Verified: 2026-07-26
Official source: https://csrc.nist.gov/projects/crypto-agility

06. NIST NCCoE - Migration to Post-Quantum Cryptography
Role: Enterprise discovery, dependency, prioritization, and migration practice.
Verified: 2026-07-26
Official source: https://www.nccoe.nist.gov/applied-cryptography/migration-to-pqc

07. IETF - Post-quantum Hybrid ECDHE-MLKEM Key Agreement for TLS 1.3
Role: Work-in-progress TLS hybrid key agreement; must be treated as an Internet-Draft.
Verified: 2026-07-26
Official source: https://datatracker.ietf.org/doc/draft-ietf-tls-ecdhe-mlkem/

08. NIST - Confidential Computing and Trusted Platforms Publications
Role: Trusted computing, attestation, roots of trust, and platform integrity reference.
Verified: 2026-07-26
Official source: https://csrc.nist.gov/topics/technologies/trusted-computing

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 Living 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; MYTHOS-CLD
Publication class	Field Guide / Canonical Living Overview Guide
Skill levels	L1 Foundational; L2 Practitioner; L3 Advanced; L4 Frontier
Tags	cryptography, post-quantum cryptography, PQC, ML-KEM, ML-DSA, SLH-DSA, crypto agility, PKI, HSM, TLS, hybrid migration
Canonical route	/library/field-guides/mythos-guide-pqc-0001/

Living publication

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