CANONICAL LIVING OVERVIEW GUIDE

# Applied Cryptography, PKI, and Key Management

An applied cryptography guide covering security goals, randomness, symmetric encryption, hashes, MACs, KDFs, public-key cryptography, signatures, key establishment, protocols, PKI, certificates, HSMs, FIPS 140-3, key lifecycle, crypto agility, testing, and implementation failures.

PERMANENT ID
MYTHOS-FG-SEC-0016

PUBLICATION CLASS
FIELD GUIDE

VERSION / STATUS
1.0.0-candidate / CANDIDATE

PERMANENT PUBLICATION IDENTITY

# Applied Cryptography, PKI, and Key Management

Document ID
MYTHOS-FG-SEC-0016

Version
1.0.0-candidate

Status
Candidate Focused Field Guide

Review date
2026-10-26

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

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

## Reader orientation

| Dimension | Engineering position |
|---|---|
| Primary domain | MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography |
| Secondary domain | MYTHOS-QTC; MYTHOS-SWE; MYTHOS-CLD |
| Audience | Security, PKI, software, cloud, platform, network, hardware, and cryptographic engineering teams. |
| Prerequisites | Basic mathematics, software or infrastructure engineering, networking, identity, and security concepts. |
| Publisher / owner | Mythos Systems / Aaron Stovall |
| Public classification | Public technical guidance |

## Expected outcomes

Select cryptographic mechanisms according to precise security properties and threat models.

Implement encryption, authentication, signatures, key establishment, and derivation without unsafe composition.

Design PKI and certificate lifecycle with reliable validation, renewal, revocation, and recovery.

Operate keys through generation, storage, use, rotation, backup, compromise, archive, and destruction.

Create crypto-agile systems prepared for algorithm, protocol, module, and post-quantum transition.

# 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 goals, models, and safe use

Define what cryptography can prove and what remains outside its protection.

Chapter operating position
Define what cryptography can prove and what remains outside its protection.

## 1.1 Confidentiality, integrity, authenticity, and nonrepudiation

Confidentiality, integrity, authenticity, and nonrepudiation must be implemented as an observable engineering mechanism, not a product label or maturity claim. Separate security properties, assumptions, actors, keys, evidence, and legal interpretations. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the security goal, primitive, key, protocol, implementation, module, lifecycle, evidence, and relying system chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

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

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

### Engineering practices

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

### Failure patterns

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

### Validation and evidence

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

## 1.2 Threat models and attack surfaces

Threat models and attack surfaces must be implemented as an observable engineering mechanism, not a product label or maturity claim. Consider passive, active, chosen-input, side-channel, implementation, endpoint, supply-chain, administrator, and future-quantum threats. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the security goal, primitive, key, protocol, implementation, module, lifecycle, evidence, and relying system chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

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

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

### Engineering practices

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

### Implementation sketch

```
cryptographic_object:
  id: signing-key-prod-01
  purpose: software-release-signing
  algorithm: approved-signature
  module: fips-140-3-validated-boundary
  state: active
  controls:
    dual_control: true
    exportable: false
    operation_requires: approved-release
  lifecycle:
    rotate_on: [scheduled, compromise, policy-change]
    backup: threshold-protected
    destroy_requires: two-authorized-operators
  evidence:
    - key-ceremony-record
    - signing-audit
    - verification-test
```

### Failure patterns

Treating threat models and attack surfaces as a product feature instead of an end-to-end engineering system.

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

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

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

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

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

### Validation and evidence

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

## 1.3 Security strength and parameter selection

Security strength and parameter selection must be implemented as an observable engineering mechanism, not a product label or maturity claim. Relate algorithms, key sizes, hash output, data lifetime, attacker resources, standards, and operational constraints. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the security goal, primitive, key, protocol, implementation, module, lifecycle, evidence, and relying system chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

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

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

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

### Engineering practices

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

### Failure patterns

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

### Validation and evidence

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

## 1.4 Composition and protocol context

Composition and protocol context must be implemented as an observable engineering mechanism, not a product label or maturity claim. Avoid inventing protocols; use established constructions, domain separation, transcript binding, identifiers, and downgrade protection. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the security goal, primitive, key, protocol, implementation, module, lifecycle, evidence, and relying system chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

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

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

### Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for composition and protocol context.

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

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

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

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

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

## Failure patterns

Treating composition and protocol context as a product feature instead of an end-to-end engineering system.

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

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

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

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

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

## Validation and evidence

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

# Chapter 01 laboratory and review

Practical laboratory
Review five cryptographic designs and identify unstated goals, unsafe composition, missing authentication, weak randomness, or endpoint assumptions.

## Laboratory evidence package

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

## Frontier extension

Extend the model to post-quantum hybrids, threshold systems, confidential computing, and privacy-enhancing cryptography.

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

# Randomness, symmetric encryption, and authenticated encryption

Generate and protect the secret material underlying cryptographic security.

Chapter operating position
Generate and protect the secret material underlying cryptographic security.

## 2.1 Entropy and random-bit generation

Entropy and random-bit generation must be implemented as an observable engineering mechanism, not a product label or maturity claim. Understand entropy sources, conditioning, DRBGs, seeding, health tests, virtualization, embedded systems, and failure. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the security goal, primitive, key, protocol, implementation, module, lifecycle, evidence, and relying system chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

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

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

### Engineering practices

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

### Failure patterns

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

### Validation and evidence

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

## 2.2 Block and stream encryption

Block and stream encryption must be implemented as an observable engineering mechanism, not a product label or maturity claim. Use approved algorithms and modes while understanding nonce, IV, padding, counters, state, and reuse risks. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the security goal, primitive, key, protocol, implementation, module, lifecycle, evidence, and relying system chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

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

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

### Engineering practices

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

### Implementation sketch

```
cryptographic_object:
  id: signing-key-prod-01
  purpose: software-release-signing
  algorithm: approved-signature
  module: fips-140-3-validated-boundary
  state: active
  controls:
    dual_control: true
    exportable: false
    operation_requires: approved-release
  lifecycle:
    rotate_on: [scheduled, compromise, policy-change]
    backup: threshold-protected
    destroy_requires: two-authorized-operators
  evidence:
    - key-ceremony-record
    - signing-audit
    - verification-test
```

### Failure patterns

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

### Validation and evidence

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

## 2.3 Authenticated encryption

Authenticated encryption must be implemented as an observable engineering mechanism, not a product label or maturity claim. Use AEAD with unique nonces, associated data, tag validation, failure handling, limits, and rekeying. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the security goal, primitive, key, protocol, implementation, module, lifecycle, evidence, and relying system chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

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

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

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

### Engineering practices

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

### Failure patterns

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

### Validation and evidence

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

## 2.4 Data-encryption architecture

Data-encryption architecture must be implemented as an observable engineering mechanism, not a product label or maturity claim. Apply envelope encryption, key hierarchy, field or object encryption, chunking, backups, metadata, and deletion. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the security goal, primitive, key, protocol, implementation, module, lifecycle, evidence, and relying system chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

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

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

### Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for data-encryption architecture.

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

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

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

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

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

## Failure patterns

Treating data-encryption architecture as a product feature instead of an end-to-end engineering system.

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

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

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

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

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

## Validation and evidence

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

# Chapter 02 laboratory and review

Practical laboratory
Implement or test an AEAD workflow, then demonstrate nonce reuse, tag failure, truncation, key confusion, and recovery handling in a safe lab.

## 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 misuse-resistant encryption, high-speed hardware, memory encryption, and quantum-safe symmetric parameter planning.

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

# Hashes, MACs, KDFs, and password protection

Use one-way functions according to their distinct purposes.

Chapter operating position
Use one-way functions according to their distinct purposes.

## 3.1 Cryptographic hash functions

Cryptographic hash functions must be implemented as an observable engineering mechanism, not a product label or maturity claim. Apply preimage, second-preimage, collision, domain separation, tree hashing, streaming, and integrity use cases. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the security goal, primitive, key, protocol, implementation, module, lifecycle, evidence, and relying system chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

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

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

### Engineering practices

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

### Failure patterns

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

### Validation and evidence

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

## 3.2 Message authentication codes

Message authentication codes must be implemented as an observable engineering mechanism, not a product label or maturity claim. Use HMAC or approved MACs with key separation, tag length, verification, replay, and protocol context. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the security goal, primitive, key, protocol, implementation, module, lifecycle, evidence, and relying system chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

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

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

### Engineering practices

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

### Implementation sketch

```
cryptographic_object:
  id: signing-key-prod-01
  purpose: software-release-signing
  algorithm: approved-signature
  module: fips-140-3-validated-boundary
  state: active
  controls:
    dual_control: true
    exportable: false
    operation_requires: approved-release
  lifecycle:
    rotate_on: [scheduled, compromise, policy-change]
    backup: threshold-protected
    destroy_requires: two-authorized-operators
  evidence:
    - key-ceremony-record
    - signing-audit
    - verification-test
```

### Failure patterns

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

### Validation and evidence

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

## 3.3 Key derivation

Key derivation must be implemented as an observable engineering mechanism, not a product label or maturity claim. Use extract-and-expand, salts, context, labels, output separation, entropy assumptions, and key hierarchy. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the security goal, primitive, key, protocol, implementation, module, lifecycle, evidence, and relying system chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

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

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

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

### Engineering practices

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

### Failure patterns

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

### Validation and evidence

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

## 3.4 Password hashing and verifier design

Password hashing and verifier design must be implemented as an observable engineering mechanism, not a product label or maturity claim. Use memory-hard derivation, salts, work factors, peppers, migration, rate limiting, and recovery. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the security goal, primitive, key, protocol, implementation, module, lifecycle, evidence, and relying system chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

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

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

### Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for password hashing and verifier design.

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

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

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

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

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

## Failure patterns

Treating password hashing and verifier design as a product feature instead of an end-to-end engineering system.

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

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

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

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

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

## Validation and evidence

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

# Chapter 03 laboratory and review

Practical laboratory
Create test vectors for hash, MAC, KDF, and password verification, including wrong context, truncated tags, weak inputs, and migration.

## 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 verifiable delay functions, memory-hard evolution, hardware-bound derivation, and privacy-preserving authentication.

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

# Public-key cryptography, signatures, and key establishment

Apply asymmetric mechanisms with correct validation and protocol binding.

Chapter operating position
Apply asymmetric mechanisms with correct validation and protocol binding.

## 4.1 RSA and elliptic-curve foundations

RSA and elliptic-curve foundations must be implemented as an observable engineering mechanism, not a product label or maturity claim. Understand keys, operations, encodings, parameter validation, padding, curves, points, and implementation constraints. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the security goal, primitive, key, protocol, implementation, module, lifecycle, evidence, and relying system chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

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

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

### Engineering practices

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

### Failure patterns

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

### Validation and evidence

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

## 4.2 Digital signatures

Digital signatures must be implemented as an observable engineering mechanism, not a product label or maturity claim. Bind message, context, identity, algorithm, representation, timestamp, and verification policy while preventing substitution. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the security goal, primitive, key, protocol, implementation, module, lifecycle, evidence, and relying system chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

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

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

### Engineering practices

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

### Implementation sketch

```
cryptographic_object:
  id: signing-key-prod-01
  purpose: software-release-signing
  algorithm: approved-signature
  module: fips-140-3-validated-boundary
  state: active
  controls:
    dual_control: true
    exportable: false
    operation_requires: approved-release
  lifecycle:
    rotate_on: [scheduled, compromise, policy-change]
    backup: threshold-protected
    destroy_requires: two-authorized-operators
  evidence:
    - key-ceremony-record
    - signing-audit
    - verification-test
```

### Failure patterns

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

### Validation and evidence

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

## 4.3 Key agreement and key establishment

Key agreement and key establishment must be implemented as an observable engineering mechanism, not a product label or maturity claim. Validate peer keys, ephemeral use, forward secrecy, KDFs, authentication, transcript, and unknown-key-share resistance. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the security goal, primitive, key, protocol, implementation, module, lifecycle, evidence, and relying system chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

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

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

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

### Engineering practices

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

### Failure patterns

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

### Validation and evidence

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

## 4.4 Hybrid and transition design

Hybrid and transition design must be implemented as an observable engineering mechanism, not a product label or maturity claim. Combine algorithms only with defined security goals, robust combiners, negotiation, downgrade resistance, and interoperability. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the security goal, primitive, key, protocol, implementation, module, lifecycle, evidence, and relying system chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

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

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

### Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for hybrid and transition design.

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

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

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

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

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

## Failure patterns

Treating hybrid and transition design as a product feature instead of an end-to-end engineering system.

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

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

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

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

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

## Validation and evidence

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

# Chapter 04 laboratory and review

Practical laboratory
Review a signing and authenticated key-establishment protocol, create test vectors, and identify validation, encoding, replay, or context failures.

## 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 threshold signatures, post-quantum KEMs and signatures, hybrid handshakes, and verifiable distributed key generation.

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

# PKI, certificates, and validation

Build and operate trust hierarchies as policy and lifecycle systems.

Chapter operating position
Build and operate trust hierarchies as policy and lifecycle systems.

## 5.1 CA hierarchy and governance

CA hierarchy and governance must be implemented as an observable engineering mechanism, not a product label or maturity claim. Design roots, intermediates, issuing CAs, profiles, policies, HSMs, ceremonies, roles, audit, and recovery. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the security goal, primitive, key, protocol, implementation, module, lifecycle, evidence, and relying system chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

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

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

### Engineering practices

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

### Failure patterns

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

### Validation and evidence

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

## 5.2 Certificate profiles

Certificate profiles must be implemented as an observable engineering mechanism, not a product label or maturity claim. Use subjects, SANs, key usage, EKU, constraints, policies, extensions, serials, lifetimes, and naming rules. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the security goal, primitive, key, protocol, implementation, module, lifecycle, evidence, and relying system chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

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

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

### Engineering practices

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

### Implementation sketch

```
cryptographic_object:
  id: signing-key-prod-01
  purpose: software-release-signing
  algorithm: approved-signature
  module: fips-140-3-validated-boundary
  state: active
  controls:
    dual_control: true
    exportable: false
    operation_requires: approved-release
  lifecycle:
    rotate_on: [scheduled, compromise, policy-change]
    backup: threshold-protected
    destroy_requires: two-authorized-operators
  evidence:
    - key-ceremony-record
    - signing-audit
    - verification-test
```

### Failure patterns

Treating certificate profiles as a product feature instead of an end-to-end engineering system.

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

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

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

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

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

### Validation and evidence

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

## 5.3 Path construction and validation

Path construction and validation must be implemented as an observable engineering mechanism, not a product label or maturity claim. Build chains, validate time, signatures, names, constraints, policies, critical extensions, and trust anchors. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the security goal, primitive, key, protocol, implementation, module, lifecycle, evidence, and relying system chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

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

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

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

### Engineering practices

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

### Failure patterns

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

### Validation and evidence

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

## 5.4 Revocation and status

Revocation and status must be implemented as an observable engineering mechanism, not a product label or maturity claim. Use CRLs, OCSP, short-lived certificates, stapling, failure policy, compromise response, and offline systems. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the security goal, primitive, key, protocol, implementation, module, lifecycle, evidence, and relying system chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

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

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

### Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for revocation and status.

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

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

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

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

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

## Failure patterns

Treating revocation and status as a product feature instead of an end-to-end engineering system.

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

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

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

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

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

## Validation and evidence

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

# Chapter 05 laboratory and review

Practical laboratory

Build a small PKI, issue certificates for users and workloads, then break naming, EKU, path, time, revocation, and policy validation.

## 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 multi-signer PKI, automated trust-store updates, short-lived workload identity, and post-quantum certificate migration.

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

# Key management, HSMs, and cryptographic modules

Control keys and modules as high-value operational assets.

Chapter operating position
Control keys and modules as high-value operational assets.

## 6.1 Key lifecycle and states

Key lifecycle and states must be implemented as an observable engineering mechanism, not a product label or maturity claim. Manage preactivation, active, suspended, deactivated, compromised, destroyed, archived, and recovered states. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the security goal, primitive, key, protocol, implementation, module, lifecycle, evidence, and relying system chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

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

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

### Engineering practices

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

### Failure patterns

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

### Validation and evidence

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

## 6.2 Generation, import, storage, and use

Generation, import, storage, and use must be implemented as an observable engineering mechanism, not a product label or maturity claim. Use HSMs, secure elements, cloud KMS, vaults, split knowledge, dual control, wrapping, policy, and attestation. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the security goal, primitive, key, protocol, implementation, module, lifecycle, evidence, and relying system chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

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

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

### Engineering practices

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

### Implementation sketch

```
cryptographic_object:
  id: signing-key-prod-01
  purpose: software-release-signing
  algorithm: approved-signature
  module: fips-140-3-validated-boundary
  state: active
  controls:
    dual_control: true
    exportable: false
    operation_requires: approved-release
  lifecycle:
    rotate_on: [scheduled, compromise, policy-change]
    backup: threshold-protected
    destroy_requires: two-authorized-operators
  evidence:
    - key-ceremony-record
    - signing-audit
    - verification-test
```

### Failure patterns

Treating generation, import, storage, and use as a product feature instead of an end-to-end engineering system.

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

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

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

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

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

### Validation and evidence

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

## 6.3 Rotation, backup, escrow, and recovery

Rotation, backup, escrow, and recovery must be implemented as an observable engineering mechanism, not a product label or maturity claim. Coordinate dependent data, certificates, services, replicas, offline systems, retention, and disaster recovery. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the security goal, primitive, key, protocol, implementation, module, lifecycle, evidence, and relying system chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

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

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

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

### Engineering practices

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

### Failure patterns

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

### Validation and evidence

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

## 6.4 FIPS 140-3 and module validation

FIPS 140-3 and module validation must be implemented as an observable engineering mechanism, not a product label or maturity claim. Understand module boundary, roles, services, algorithms, self-tests, physical security, configuration, and validation scope. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the security goal, primitive, key, protocol, implementation, module, lifecycle, evidence, and relying system chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

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

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

### Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for fips 140-3 and module validation.

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

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

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

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

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

## Failure patterns

Treating fips 140-3 and module validation as a product feature instead of an end-to-end engineering system.

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

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

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

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

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

## Validation and evidence

| Method | Expected evidence |
| --- | --- |
| Examine | Requirements, architecture, identities, data, algorithms, policy, configuration, ownership, and dependencies for fips 140-3 and module validation. |
| Observe | Active state, applied policy, raw events, telemetry, artifacts, cryptographic results, resources, and endpoint behavior. |
| Test | Expected behavior, prohibited behavior, malformed inputs, dependency loss, partial failure, scale, recovery, and rollback. |
| Measure | Availability, latency, detection quality, evidence completeness, cryptographic cost, risk, resource use, and mission outcome. |
| Reconcile | Intended state against observed security, identity, application, cloud, cryptographic, data, and evidence records. |

# Chapter 06 laboratory and review

Practical laboratory
Create a key-management plan and ceremony for signing and encryption keys, then test compromise, HSM loss, restore, rotation, and audit.

## 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 threshold HSM services, confidential key release, remote attestation, and post-quantum module validation.

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

# Protocols, applications, and implementation safety

Use cryptography inside complete systems with correct negotiation, parsing, and error handling.

Chapter operating position
Use cryptography inside complete systems with correct negotiation, parsing, and error handling.

## 7.1 TLS and secure channels

TLS and secure channels must be implemented as an observable engineering mechanism, not a product label or maturity claim. Validate peer identity, versions, cipher suites, certificates, key exchange, resumption, early data, alerts, and proxies. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the security goal, primitive, key, protocol, implementation, module, lifecycle, evidence, and relying system chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

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

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

### Engineering practices

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

### Failure patterns

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

### Validation and evidence

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

## 7.2 Storage, messaging, and signing systems

Storage, messaging, and signing systems must be implemented as an observable engineering mechanism, not a product label or maturity claim. Apply encryption and signatures to files, databases, backups, messages, artifacts, logs, and updates. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the security goal, primitive, key, protocol, implementation, module, lifecycle, evidence, and relying system chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

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

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

### Engineering practices

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

### Implementation sketch

```
cryptographic_object:
  id: signing-key-prod-01
  purpose: software-release-signing
  algorithm: approved-signature
  module: fips-140-3-validated-boundary
  state: active
  controls:
    dual_control: true
    exportable: false
    operation_requires: approved-release
  lifecycle:
    rotate_on: [scheduled, compromise, policy-change]
    backup: threshold-protected
    destroy_requires: two-authorized-operators
  evidence:
    - key-ceremony-record
    - signing-audit
    - verification-test
```

### Failure patterns

Treating storage, messaging, and signing systems as a product feature instead of an end-to-end engineering system.

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

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

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

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

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

### Validation and evidence

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

## 7.3 Side channels and implementation attacks

Side channels and implementation attacks must be implemented as an observable engineering mechanism, not a product label or maturity claim. Address timing, cache, power, fault, memory, branches, error oracles, parsing, and compiler behavior. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the security goal, primitive, key, protocol, implementation, module, lifecycle, evidence, and relying system chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

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

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

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

### Engineering practices

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

### Failure patterns

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

### Validation and evidence

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

## 7.4 Testing and validation

Testing and validation must be implemented as an observable engineering mechanism, not a product label or maturity claim. Use known-answer, negative, malformed, interoperability, fuzz, boundary, performance, and failure tests. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the security goal, primitive, key, protocol, implementation, module, lifecycle, evidence, and relying system chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

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

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

### Engineering practices

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

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

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

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

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

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

## Failure patterns

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

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

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

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

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

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

## Validation and evidence

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

# Chapter 07 laboratory and review

Practical laboratory
Build a protocol test plan covering negotiation, certificate validation, downgrade, malformed inputs, side-channel assumptions, and failure responses.

## 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 verified cryptographic libraries, constant-time languages, hardware isolation, and proof-carrying 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

# Crypto agility, inventory, migration, and governance

Prepare for algorithm deprecation, compromise, regulation, and quantum transition.

Chapter operating position
Prepare for algorithm deprecation, compromise, regulation, and quantum transition.

## 8.1 Cryptographic inventory

Cryptographic inventory must be implemented as an observable engineering mechanism, not a product label or maturity claim. Find algorithms, protocols, libraries, modules, keys, certificates, data, firmware, partners, hardware, and hidden dependencies. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the security goal, primitive, key, protocol, implementation, module, lifecycle, evidence, and relying system chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

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

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

### Engineering practices

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

### Failure patterns

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

### Validation and evidence

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

## 8.2 Policy and agility architecture

Policy and agility architecture must be implemented as an observable engineering mechanism, not a product label or maturity claim. Separate algorithm identifiers, parameters, negotiation, storage, interfaces, configuration, testing, and rollout from business logic. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the security goal, primitive, key, protocol, implementation, module, lifecycle, evidence, and relying system chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

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

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

### Engineering practices

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

### Implementation sketch

```
cryptographic_object:
  id: signing-key-prod-01
  purpose: software-release-signing
  algorithm: approved-signature
  module: fips-140-3-validated-boundary
  state: active
  controls:
    dual_control: true
    exportable: false
    operation_requires: approved-release
  lifecycle:
    rotate_on: [scheduled, compromise, policy-change]
    backup: threshold-protected
    destroy_requires: two-authorized-operators
  evidence:
    - key-ceremony-record
    - signing-audit
    - verification-test
```

### Failure patterns

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

### Validation and evidence

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

## 8.3 Migration and coexistence

Migration and coexistence must be implemented as an observable engineering mechanism, not a product label or maturity claim. Plan compatibility, hybrid periods, trust updates, re-encryption, re-signing, old data, rollback, and retirement. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the security goal, primitive, key, protocol, implementation, module, lifecycle, evidence, and relying system chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

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

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

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

### Engineering practices

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

### Failure patterns

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

### Validation and evidence

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

## 8.4 Governance and review

Governance and review must be implemented as an observable engineering mechanism, not a product label or maturity claim. Assign crypto authority, exceptions, standards, validation, incident response, supplier requirements, and review triggers. The design should name authoritative inputs, identities, state transitions, dependencies, cryptographic or policy boundaries, limits, owners, telemetry, failure behavior, and acceptance criteria.

This guide traces the subject through the security goal, primitive, key, protocol, implementation, module, lifecycle, evidence, and relying system chain. A configured integration, successful API response, established cryptographic session, green dashboard, or closed ticket is not sufficient proof. Intended state must be reconciled with applied state, negative tests, degraded behavior, raw evidence, recovery, and the resulting security or mission outcome.

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

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

### Engineering practices

Define authority, ownership, scope, dependencies, limits, and acceptance criteria for governance and review.

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

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

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

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

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

## Failure patterns

Treating governance and review as a product feature instead of an end-to-end engineering system.

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

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

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

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

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

## Validation and evidence

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

# Chapter 08 laboratory and review

Practical laboratory
Create a crypto inventory and migration plan for one service, including TLS, PKI, stored data, backups, code signing, HSMs, and partners.

## Laboratory evidence package

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

## Frontier extension

Develop a cryptographic control plane that continuously reconciles inventory, policy, standards, and 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.

# 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 - SP 800-57 Part 1 Rev. 5 - Recommendation for Key Management
Role: Final key-management principles and lifecycle guidance.
Verified: 2026-07-26
Official source: https://csrc.nist.gov/pubs/sp/800/57/pt1/r5/final

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

03. NIST - FIPS 140-3 - Security Requirements for Cryptographic Modules
Role: Security requirements and validation foundation for cryptographic modules.
Verified: 2026-07-26
Official source: https://csrc.nist.gov/pubs/fips/140-3/final

04. NIST - FIPS 186-5 - Digital Signature Standard
Role: Approved classical digital-signature algorithms and requirements.
Verified: 2026-07-26
Official source: https://csrc.nist.gov/pubs/fips/186-5/final

05. IETF / RFC Editor - RFC 5280 - Internet X.509 PKI Certificate and CRL Profile
Role: X.509 certificate and certification-path validation profile.
Verified: 2026-07-26
Official source: https://www.rfc-editor.org/info/rfc5280

06. IETF / RFC Editor - RFC 8017 - PKCS #1: RSA Cryptography Specifications Version 2.2
Role: RSA encryption and signature encoding specification.
Verified: 2026-07-26
Official source: https://www.rfc-editor.org/info/rfc8017

07. IETF / RFC Editor - RFC 8446 - The Transport Layer Security Protocol Version 1.3
Role: TLS 1.3 protocol specification.
Verified: 2026-07-26
Official source: https://www.rfc-editor.org/info/rfc8446

08. NIST - SP 800-56A Rev. 3 - Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography
Role: Classical discrete-logarithm key-establishment guidance.
Verified: 2026-07-26
Official source: https://csrc.nist.gov/pubs/sp/800/56/a/r3/final

09. NIST - SP 800-56C Rev. 2 - Recommendation for Key-Derivation Methods in Key-Establishment Schemes
Role: Key derivation for key-establishment schemes.
Verified: 2026-07-26
Official source: https://csrc.nist.gov/pubs/sp/800/56/c/r2/final

## Authority application and refresh triggers

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

# Limitations, freshness, and revision control

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

## Revision record

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

## Search and relationship metadata

| Dimension | Engineering position |
| --- | --- |
| Primary domain | MYTHOS-SEC - Cybersecurity, Security Engineering & Cryptography |
| Secondary domain | MYTHOS-QTC; MYTHOS-SWE; MYTHOS-CLD |
| Publication class | Field Guide / Canonical Living Overview Guide |
| Skill levels | L1 Foundational; L2 Practitioner; L3 Advanced; L4 Frontier |
| Tags | applied cryptography, PKI, key management, HSM, FIPS 140-3, TLS, certificates, AEAD, digital signatures, crypto agility |
| Canonical route | /library/field-guides/mythos-fg-sec-0016/ |

Living publication

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